MLOOF: MULTI-LEVEL ONLINE OFFLOAD FRAMEWORK - ARCABOUÇO DE DESCARREGAMENTO DE PROCESSAMENTO MULTI-NÍVEL

LEANDRO NOMAN FERREIRA

MLOOF: MULTI-LEVEL ONLINE OFFLOAD FRAMEWORK - ARCABOUÇO DE DESCARREGAMENTO DE PROCESSAMENTO MULTI-NÍVEL

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: José Marcos Silva Nogueira
Coorientador: Daniel Fernandes Macedo

Belo Horizonte

Novembro de 2020

© 2020, Leandro Noman Ferreira.
Todos os direitos reservados

Ferreira, Leandro Noman.

F383m MLOOF: [manuscrito] multi-level online offload framework / Leandro Noman Ferreira. - 2020.
xxii, 66 f. il.

Orientador: José Marcos Silva Nogueira
Coorientador: Daniel Fernandes Macedo

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f.63-66.

1. Computação – Teses. 2. Internet das coisas – Teses. 3. Sistemas de comunicação móvel – Teses. 4. Computação em nuvem - Teses. I. Nogueira, José Marcos Silva. .II. Macedo, Daniel Fernandes. III. Universidade Federal de Minas Gerais; Instituto de Ciências Exatas, Departamento de Ciência da Computação. IV. Título.

CDU 519.6*22(043)

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa
CRB 6ª Região nº 1510

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

MLOOF: Multi-level Online Offload Framework

LEANDRO NOMAN FERREIRA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. JOSÉ MARCOS SILVA NOGUEIRA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. DANIEL FERNANDES MACEDO - Coorientador
Departamento de Ciência da Computação - UFMG

PROFA. PRISCILA AMÉRICA SOLÍS MENDEZ BARRETO
Departamento de Ciência da Computação - UNB

PROF. VINÍCIUS FERNANDES SOARES MOTA
Departamento de Informática - UFES

PROF. ÍTALO FERNANDO SCOTÁ CUNHA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 23 de Novembro de 2020.

Agradecimentos

Os autores agradecem às agências CAPES, CNPq e Fapemig pelo apoio, seja na forma de bolsas de estudo ou auxílio a pesquisa. Agradecimentos também ao aluno de iniciação científica, Gustavo Guedes de Azevedo Barbosa, que projetou e criou o sistema de medição de energia, essencial para a realização dos experimentos desse trabalho.

Resumo

O termo *offloading* indica a ação de se alterar o local de processamento de uma atividade computacional. Os objetivos de se usar *offloading* são reduzir o tempo de processamento de aplicativos, diminuir o consumo energético dos dispositivos e eventualmente possibilitar execução de tarefas que não seriam possíveis em dispositivos de recursos reduzidos. Este trabalho apresenta um arcabouço de *offloading* chamado MLOOF (Multi-Level Online Offloading Framework). O MLOOF possui uma arquitetura de três camadas composta pelos dispositivos, *cloudlets* (servidores intermediários) e servidores na nuvem. A adição das *cloudlets* possibilita o aumento da vazão da rede e a redução da latência entre dispositivo e servidor por estarem próximas aos dispositivos. O código rodando no dispositivo possui um motor de decisão que escolhe, ao iniciar um método, onde o método deve ser executado, se em um dos servidores ou localmente. A decisão leva em consideração previsões de tempo de processamento nos diferentes ambientes de execução e consumo energético do próprio dispositivo. O sistema foi avaliado de forma experimental em um ambiente semi-controlado, utilizando um *testbed* e servidores dedicados na nuvem, e utilizando um simulador. Os resultados apontam que a estratégia do arcabouço de *offloading* em três níveis consegue atingir os objetivos de redução de tempo de processamento e de consumo de energia, trazendo grandes ganhos, principalmente quando utilizado por dispositivos com menos recursos. Nos testes realizados, o tempo de execução utilizando as *cloudlets* chega a ser 49% menor do que a execução na nuvem e o consumo energético ao executar o código remotamente se mantém constante, enquanto que ao executar localmente o consumo é exponencial.

Abstract

The term offloading indicates the action of changing the processing location of a computational activity. The purpose of using offloading is to reduce the processing time of applications, reduce the power consumption of the devices and eventually enable the execution of tasks that would not be possible on devices with reduced resources. This work presents an offloading framework called MLOOF (Multi-Level Online Offloading Framework). MLOOF has a three-level architecture composed of devices, cloudlets (intermediate servers) and cloud servers. The addition of cloudlets makes it possible to increase network throughput and reduce latency between device and server as they are closer to the devices. The code running on the device has a decision engine that chooses, when starting a method, where the method should be executed, whether on one of the servers or locally. The decision takes into account processing time prediction in the different execution environments and energy consumption of the device. The system was evaluated experimentally in a semi-controlled environment, using a test-bed and dedicated servers in the cloud, and using a simulator. The results show that the three-level offloading framework strategy achieves the goals of reducing processing time and energy consumption, bringing great gains, especially when used by devices with less computational resources. In our experiments, the execution time using the cloudlets is up to 49% less than the execution on the cloud and the energy consumption when executing the code remotely remains constant, while when executing locally the consumption is exponential.

Lista de Figuras

Lista de Tabelas

Lista de acrônimos

IoT = Internet of Things (Internet das Coisas)
RFID = Radio Frequency IDentification (Identificação por Radiofrequência)
M2M = Machine to Machine (Máquina para Máquina)
D2D = Device to Device (Dispositivo para Dispositivo)
ULOOF = User-level Online Offloading Framework
MLOOF = Multi-level Online Offload Framework
WAN = Wide Area Network
LAN = Local Area Network
API = Application Programming Interface (Interface de Programação de Aplicação)
CPU = Central Processing Unit (Unidade Central de Processamento)
UDP = User Datagram Protocol
URL = Uniform Resource Locator
HTTP = HyperText Transfer Protocol (Protocolo de Transferência de Hipertexto)
IP = Internet Protocol (Protocolo de Internet)
USB = Universal Serial Bus (Porta Serial Universal)
PC = Personal Computer (Computador Pessoal)
AWS = Amazon Web Services (Serviços Web da Amazon)

Sumário

Capítulo 1

Introdução

A Internet das Coisas (IoT) é um paradigma que ganhou muita repercussão no passado recente e continua crescendo [Cisco Systems, 2020]. A ideia básica é a presença de objetos ou coisas - como etiquetas RFID (Radio Frequency IDentification), sensores, atuadores, aparelhos celulares, etc - que tem a capacidade de interagir uns com os outros e cooperar com os seus vizinhos para atingir objetivos comuns [Atzori et al., 2010].

Segundo [Cisco Systems, 2020], a perspectiva é que existirão 29.3 bilhões de dispositivos conectados à internet em 2023 (3.6 dispositivos por pessoa do planeta), um aumento de mais de 59% do número de dispositivos conectados em 2018. Desse número, metade será de conexões máquina para máquina - *machine-to-machine* (M2M, que também são chamadas de conexões IoT [Cisco Systems, 2020]).

Dispositivos móveis tem poder de processamento e bateria limitados [Mao et al., 2016] por natureza, o que gera uma grande preocupação em aumentar a eficiência energética dos aparelhos e sua capacidade computacional [Kumar et al., 2013].

Algumas soluções foram propostas para suprir a necessidade de processamento e, ao mesmo tempo, melhorar a eficiência energética desses dispositivos [Cuervo et al., 2010], [Kemp et al., 2010], [Chun et al., 2011]. Grande parte dessas propostas envolve a utilização de Computação em Nuvem - ou *Cloud Computing* - para realizar o processamento que seria feito no próprio dispositivo - seja esse dispositivo um *smartphone* ou dispositivo IoT. *Cloud Computing* é um paradigma de computação que permite o acesso facilitado a recursos computacionais compartilhados através de uma rede de computadores de forma ubíqua, conveniente e sobre demanda, que possam ser reservados e liberados de forma rápida com mínimo esforço administrativo ou interação com o provedor de serviços [Mell et al., 2011]. Utilizar a Computação em Nuvem pode ser vantajosa quando houver diminuição do tempo de execução da tarefa a ser realizada

ou diminuição do consumo energético do dispositivo quando comparado à execução no próprio dispositivo. Dependendo do poder de processamento e quantidade de memória do dispositivo e das exigências computacionais da tarefa em questão, executar a tarefa no dispositivo pode ser mais rápido e pode consumir menos energia do que transferir a execução para um servidor na nuvem.

Mesmo a utilização da computação em nuvem pode não ser suficiente para algumas aplicações que tenham restrições muito grandes no tempo de execução - nas quais o processamento deve ser feito em tempo real, por exemplo. Altas latências provenientes da WAN (Wide Area Network) podem se mostrar um obstáculo fundamental para tais aplicações. A ideia de computação em borda (ou *edge computing*) surge como uma forma de trazer recursos de computação e armazenamento para a borda da rede, mais próximos aos dispositivos finais [Satyanarayanan, 2017]. Estes servidores na borda da rede, comumente chamados de *cloudlets* representam uma camada intermediária em uma hierarquia de três camadas: dispositivo IoT – *cloudlet* – *cloud*. Dessa forma é possível atender as demandas por respostas interativas e em tempo real com baixa latência e alta largura de banda utilizando o acesso sem fio à *cloudlet* [Satyanarayanan et al., 2009].

Podemos utilizar as tecnologias em nuvem para alavancar as capacidades de dispositivos com recursos limitados, como *smartphones* e dispositivos IoT. São utilizadas técnicas para transferência do processo de computação de um dispositivo para outro - de um computador pessoal ou de um *smartphone* para a nuvem, por exemplo. Dessa forma, é possível economizar energia dos dispositivos mais limitados e até mesmo diminuir o tempo de execução das aplicações, tornando viável a execução de programas que requerem alta intensidade de computação que, à primeira vista, não poderiam executar em dispositivos móveis [Kumar & Lu, 2010]. Tais técnicas são chamadas de *offloading*.

1.1 Motivação

Um dos grandes problemas em sistemas de IoT e em dispositivos móveis é a quantidade limitada de recursos [Mao et al., 2016], [Satyanarayanan, 1996]. Seja por falta de memória, poder de processamento ou restrições impostas pela bateria dos dispositivos, algumas aplicações são complexas demais ou mesmo inviáveis de serem executadas nesses ambientes. Mesmo quando os dispositivos conseguem executar os programas, eles podem ser forçados a rodar mais lentamente [Kumar & Lu, 2010] ou gastar muita carga da bateria.

A quantidade de trabalhos que utilizam soluções de *offloading* para dispositivos

móveis [Kumar et al., 2013] tem aumentado na literatura, sendo a evolução da qualidade das redes móveis um dos fatores que possibilitam o sucesso de tais soluções [Bhalla & Bhalla, 2010]. A evolução dos dispositivos móveis possibilita a realização de tarefas mais complexas que exigem maior poder de processamento. A melhora na qualidade e capacidade das redes sem fio e a maior preocupação com a duração das baterias de dispositivos móveis criam um ambiente favorável para a aplicação de técnicas de *offloading*.

[El Baz, 2014] apresenta algumas aplicações na área de internet das coisas que estão fortemente conectadas a problemas de computação de alto desempenho. A primeira diz respeito à administração de um edifício inteligente, onde milhares de sensores coletam informação em tempo real e soluções de problemas combinatórios devem ser empregadas para o controle de todo o sistema. Outra aplicação é relacionada a logística, no qual dispositivos móveis são utilizados para informar sobre entregas bem efetuadas ou incidentes, como falhas em motores dos veículos de entrega e tráfego lento. Os dispositivos móveis também iniciam computações relacionadas a problemas de roteamento para tratarem de incidentes em tempo real, que devem ser realizadas em uma estrutura computadorizada mais robusta.

Um outro exemplo de aplicação seria uma câmera de segurança, com poder de processamento insuficiente para uma certa aplicação. Se a câmera não conseguir comprimir o vídeo em tempo real, as imagens poderiam ser enviadas a uma *cloudlet* que faria a compressão dos dados e enviaria a informação já comprimida a um servidor na nuvem. Este seria responsável pelo armazenamento. Além disso, algoritmos de visão computacional podem ser executados na *cloudlet* permitindo que a aplicação de técnicas de reconhecimento de imagens e/ou aprendizado de máquina sejam empregados em tempo real. [Motlagh et al., 2017] apresenta um trabalho similar no qual veículos aéreos autônomos capturam imagens que são descarregadas para a borda da rede onde algoritmos de reconhecimento facial são executados.

Em todas essas situações uma solução de *offloading* pode aumentar a responsividade da aplicação ou diminuir o gasto energético dos dispositivos.

1.2 Componentes Básicos

Em todo o texto serão utilizadas palavras e expressões referentes aos componentes do sistema. A lista a seguir explicita algumas destas palavras para auxiliar a compreensão e evitar ambiguidades:

- **Usuário:** O usuário do nosso arcabouço é o programador que utiliza o nosso

sistema no desenvolvimento de suas aplicações.

- **Aplicação e aplicativo:** Software desenvolvido utilizando nossa solução. A aplicação ou aplicativo é desenvolvida para rodar nos dispositivos móveis (sejam dispositivos IoT ou *smartphones*) mas, para otimizar o processo de *offloading*, também deve estar presente nos servidores remotos.

- **Dispositivo, cliente e consumidor:** Dispositivo físico que executa a aplicação descrita acima.

- **Termos comuns em inglês e suas traduções:** Alguns termos em inglês e suas traduções são utilizadas ao decorrer desta dissertação de forma indiscriminada. Alguns deles são: *offloading* e descarregamento; *smartphones* e celulares; *IoT* e internet das coisas; *cloud* e nuvem.

1.3 Objetivos

O objetivo geral do trabalho é estudar soluções de *offloading* para a internet das coisas e para *smartphones*, e desenvolver um sistema funcional que consiga diminuir o consumo energético e o tempo de execução de tarefas nesses dispositivos.

A arquitetura do sistema é organizada em várias camadas. A Figura 1.1 ilustra essa arquitetura com um exemplo simples. Os dispositivos móveis se conectam a uma *cloudlet* que está logicamente próxima aos dispositivos, por exemplo em uma rede local. A *cloudlet* pode realizar a computação para os dispositivos e se comunicar com os servidores na nuvem para armazenar e sincronizar dados, obter informações salvas anteriormente ou até executar a computação na nuvem. Podem existir momentos em que nenhuma *cloudlet* esteja disponível. Nesse caso os dispositivos se conectam diretamente aos servidores na nuvem.

Além da concepção do sistema descrito, um objetivo secundário do trabalho é o estudo de um algoritmo de decisão distribuído. Na maioria das soluções propostas para *offloading* de processamento, toda a decisão é realizada no próprio dispositivo. A ideia de um sistema distribuído é que os servidores armazenem as informações geradas por vários clientes e consigam criar regras de decisão de forma estática para serem enviadas e utilizadas pelos dispositivos.

Por fim, é objetivo do trabalho avaliar a solução proposta em ambiente real e simulações. A avaliação em ambiente real elimina dúvidas em relação à modelagem de um sistema simulado e é afetada por variações inerentes aos experimentos reais, como a qualidade de conexão e variações de performance do hardware físico. Por outro lado,

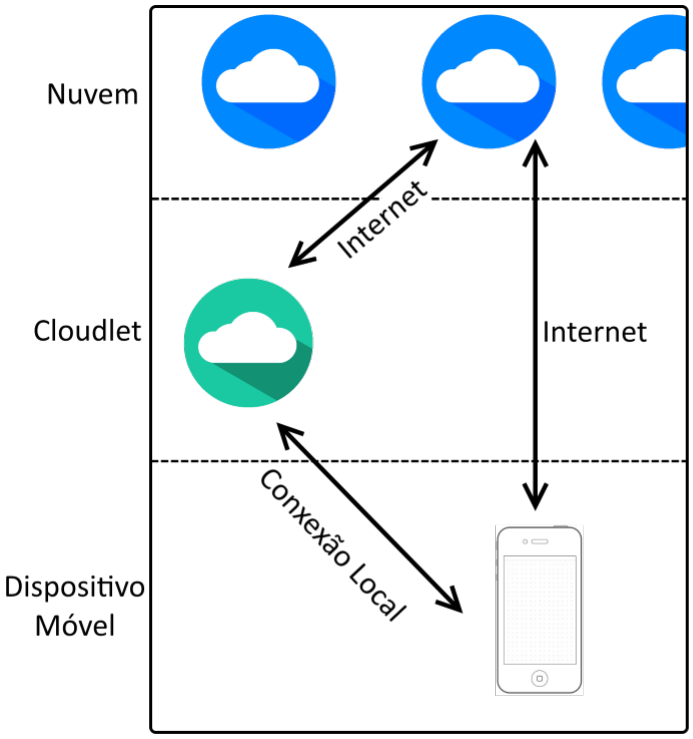

Figura 1.1. Ilustração da arquitetura de comunicação entre dispositivos, *cloudlet* e *cloud*.

o experimento real é limitado em questão de disponibilidade de dispositivos e falta de controle sobre a rede. As simulações conseguem suprir essa necessidade de avaliação de outras variáveis como a escalabilidade e a variação das características dos dispositivos e da rede.

1.4 Contribuições

Este trabalho apresenta a concepção de uma solução de *offloading* para IoT e *smartphones* utilizando uma infraestrutura de nuvem multi-nível, com servidores em nuvem e *cloudlets* para diminuir a latência e diminuir o tempo de resposta do sistema. Este trabalho é, em certo sentido, uma evolução do trabalho ULOOF (*User-level Online Offloading Framework*) [Neto et al., 2018], que trata de *offloading* em estrutura de dois níveis. A proposta adiciona um nível à estrutura do sistema (dispositivo - *cloudlet* - nuvem), e pode ser utilizado com dispositivos IoT.

O trabalho executado traz as seguintes contribuições:

1. Proposta e implementação de uma arquitetura multi-camadas composta por dis-

positivos móveis, servidores em *cloudlets* e servidores na nuvem. Servidores e dispositivos cooperam para a execução de aplicações, que inicialmente seriam executadas exclusivamente nos dispositivos móveis, diminuindo o tempo de execução das aplicações e o consumo energético dos dispositivos.

2. Estudo de melhorias para algoritmos de decisão de *offloading*.

3. Avaliação da solução proposta dividida em três partes: por experimento em ambiente real; Por experimento utilizando um *testbed* como servidor intermediário; Por simulação, para verificação de escalabilidade da solução em dispositivos diferentes aos utilizados nos outros experimentos.

1.5 Organização do Documento

Este documento está organizado em cinco capítulos. O primeiro consiste de uma introdução que inclui conceitos básicos importantes e os objetivos do trabalho. O segundo capítulo aprofunda nos conceitos essenciais e traz uma análise da literatura, apontando os principais trabalhos relacionados. O terceiro capítulo apresenta a proposta, incluindo a arquitetura multi-nível, a descrição de seus componentes e a implementação. O quarto capítulo apresenta as três categorias de experimentos, bem como seus resultados e avaliações. Por fim, o quinto capítulo apresenta nossas conclusões e trabalhos futuros.

Capítulo 2

Conceitos e Trabalhos Relacionados

Este capítulo está dividido em três partes. A primeira parte apresenta os conceitos mais importantes para o entendimento deste trabalho, incluindo os conceitos de Internet das Coisas (IoT), *Cloud*, *Cloudlet* e *offloading*. A segunda parte consiste de uma revisão da literatura, apresentando e discutindo os trabalhos relacionados ao trabalho proposto. A terceira parte apresenta o *framework* de *offloading* ULOOF, que será utilizado como base para este trabalho.

2.1 Conceitos

2.1.1 Internet das Coisas

A Internet das Coisas é um novo paradigma que vem se consolidando nos últimos anos e está mudando a forma como vemos e fazemos uso da internet. A ideia básica é a presença de objetos ou coisas - como etiquetas RFID (Radio Frequency IDentification), sensores, atuadores, aparelhos celulares, etc - que tem a capacidade de interagir uns com os outros e cooperar com os seus vizinhos para atingir objetivos comuns [Atzori et al., 2010].

De acordo com [Miorandi et al., 2012], podemos criar a noção de objetos inteligentes, que são as "coisas" que compõem a Internet das Coisas. Esses objetos inteligentes são complementos para as entidades que fazem parte da internet tradicional (hosts, terminais, roteadores, etc). De um ponto de vista conceitual, a IoT se baseia em três pilares, relacionados à capacidade dos objetos inteligentes: (i) serem identificáveis. Qualquer coisa deve ser capaz de se identificar ou ser identificada; (ii) se comunicarem, os objetos devem ter a capacidade de comunicação entre si e com unidades; e (iii) interagirem, seja com outras coisas, formando uma rede de objetos interconectados, ou

mesmo com o usuário final. [Miorandi et al., 2012] definem de outra forma algumas características necessárias de um objeto inteligente, sendo elas:

- Possuir forma física e as características físicas associadas, como tamanho, forma, etc.
- Possuir um conjunto mínimo de funcionalidades de comunicação, como a habilidade de receber e responder mensagens.
- Ter associado a ele, ao menos um nome e um endereço. O nome é uma descrição do objeto que possa ser lida e interpretada por um ser humano, enquanto que o endereço é uma informação que possa ser lida por uma máquina e que possa ser utilizada para localização e comunicação com o objeto.
- Possuir capacidades básicas de computação, sejam elas simples como interpretar uma mensagem recebida, ou complexas como a realização de computações.
- Pode possuir mecanismos para sensorear fenômenos físicos, como medir temperatura e intensidade de luz.
- Pode possuir mecanismos para atuar no mundo físico, como o acionamento de um interruptor.

2.1.2 Cloud Computing

Cloud Computing - ou Computação em Nuvem - é um paradigma de computação que permite o acesso facilitado a recursos computacionais compartilhados através de uma rede de computadores de forma ubíqua, conveniente e sobre demanda, que possam ser reservados e liberados de forma rápida com mínimo esforço administrativo ou interação com o provedor de serviços [Mell et al., 2011].

Ainda de acordo com os mesmos autores, o modelo de *Cloud Computing* é composto de cinco características essenciais e três modelos de serviço. As características essenciais são:

1. Auto-serviço sob demanda: O consumidor, ou cliente, pode reservar, de forma unilateral, recursos ou capacidades de computação tais como tempo de utilização de servidores e armazenamento em rede sob demanda, sem a necessidade de interação humana com o provedor de serviços.
2. Amplo acesso à rede: Os recursos são disponibilizados através de uma rede de computadores e podem ser acessados por mecanismos comuns que promovam o

uso por diferentes plataformas utilizadas pelo cliente - como telefones celulares, tablets, laptops e estações de trabalho.

3. Agrupamento dinâmico de recursos: Os recursos oferecidos pelo provedor podem ser agrupados e divididos entre vários clientes, com diferentes recursos físicos e virtuais alocados e realocados dinamicamente, de acordo com as demandas dos clientes. A localização geográfica desses recursos não é necessariamente de controle ou mesmo de conhecimento dos clientes, mas pode ser especificada em diferentes níveis de abstração - como país, estado ou datacenter. Exemplos de recursos incluem armazenamento, processamento, memória e banda de comunicação.

4. Rápida realocação: Os recursos podem ser rapidamente realocados, em alguns casos automaticamente, para se adequarem de acordo com a necessidade. Do ponto de vista do cliente, os recursos podem ser ilimitados e podem ser alocados em qualquer quantidade a qualquer momento.

5. Serviço mensurável: Os sistemas em nuvem podem controlar automaticamente a utilização de recursos de acordo com alguma métrica de capacidade - normalmente alguma forma de pagamento-por-uso ou recarga-por-uso do recurso - em algum nível de abstração que seja apropriado ao tipo de serviço - por exemplo, armazenamento, processamento, largura de rede e número de usuários ativos. A utilização dos recursos pode ser monitorada, controlada e reportada de forma transparente por ambos provedor e cliente do serviço utilizado.

Os três modelos de serviço são:

1. Software como Serviço (SaaS): A capacidade provida aos usuários é a utilização de aplicações rodando em uma infraestrutura em nuvem. As aplicações são acessíveis aos clientes através de uma interface simples - como um navegador web - ou através de um programa. O consumidor não tem controle sobre a infraestrutura utilizada pela nuvem e nem sobre as capacidades individuais da aplicação, com a possível exceção de configurações específicas para clientes.

2. Plataforma como Serviço (PaaS): A capacidade provida aos usuários é utilizar a infraestrutura em nuvem para rodar aplicações criadas ou adquiridas pelo consumidor, desde que haja suporte pelo provedor para executá-las. O consumidor não tem controle sobre a infraestrutura utilizada pela nuvem, mas tem controle sobre as aplicações.

3. Infraestrutura como Serviço (IaaS): A capacidade provida aos usuários é a utilização de armazenamento, processamento, redes e outros recursos computacionais fundamentais onde o consumidor pode rodar softwares arbitrários, que pode incluir sistemas operacionais e aplicações. O consumidor não tem controle sobre a infraestrutura utilizada pela nuvem, mas tem controle sobre sistemas operacionais, armazenamento, aplicações instaladas e possivelmente controle limitado sobre elementos de rede - como firewalls por exemplo.

2.1.3 *Cloudlet*

O hardware de dispositivos móveis está em constante evolução, mas estamos conformados com a ideia que o hardware de dispositivos móveis são necessariamente pobres em recursos se comparados ao hardware de clientes estáticos ou de servidores [Satyanarayanan, 1996]. Essa limitação em recursos é um grande obstáculo para aplicações que focam em aumentar a cognição humana, por se tratarem de aplicações que requerem poder de processamento e energia acima das capacidades providas pelo hardware dos dispositivos móveis, mesmo havendo hoje dispositivos móveis com grande capacidade de processamento, já que as novas aplicações demandam cada vez mais recursos.

Uma solução para a falta de recursos em dispositivos móveis é a utilização de serviços baseados em computação em nuvem. O dispositivo pode executar tarefas que requerem muitos recursos em um servidor de alto desempenho enquanto o aplicativo que executa no dispositivo provê uma interface entre o próprio dispositivo e o servidor via internet. Infelizmente, altas latências das redes WAN (Wide Area Network) são um obstáculo fundamental. [Lagar-Cavilla et al., 2007] mostrou que a latência impacta negativamente a utilização de aplicações interativas, mesmo quando a banda de comunicação é suficiente.

Computação em borda, ou *edge computing*, é um paradigma no qual recursos computacionais e de armazenamento são trazidos para a borda da rede para prover conexão de melhor qualidade aos dispositivos. Servidores que proveem os recursos computacionais na borda da rede são chamados de *cloudlets* e podem ser vistos como a miniaturização de data centers, representando a camada intermediária em uma arquitetura de três camadas: dispositivo – *cloudlet* – *cloud*. Dessa forma, é possível atender às demandas por respostas interativas e em tempo real com baixa latência e alta largura de banda utilizando o acesso sem fio à *cloudlet* [Satyanarayanan et al., 2009]. Para isso, a proximidade física da *cloudlet* é essencial, e o tempo de resposta para aplicações rodando na *cloudlet* deve ser de apenas alguns milissegundos. Ainda de acordo com [Satyanarayanan et al., 2009], a *cloudlet* se diferencia da *cloud* em alguns

pontos-chave. A principal diferença é que a *cloudlet* retém apenas o *soft state*, como dados copiados para uma cache ou código que está presente em outro lugar, fazendo com que a perda da *cloudlet* ou da conexão com ela não seja catastrófica. A *cloudlet* também é autogerida, necessitando de pouco mais que energia, conexão com a internet e controle de acesso para sua instalação. Internamente, uma *cloudlet* se assemelha a um cluster de computadores, com conectividade gigabit e alta largura de banda em sua rede sem fio LAN (Local Area Network).

2.1.4 Offloading

O termo *offloading*, no contexto deste documento, refere-se à ação de mudar o local onde um processo de computação é feito - de um computador pessoal ou um *smartphone* para a nuvem, por exemplo. As razões para isso podem incluir economia de energia e diminuição no tempo de execução das aplicações. De acordo com [Kumar & Lu, 2010], não é possível utilizar algumas aplicações em sistemas móveis devido à alta intensidade de computação requerida por elas, fazendo com que a aplicação tenha que ser executada na nuvem. Outras aplicações como recuperação de imagens, reconhecimento de fala, jogos e navegação podem ser executados em dispositivos móveis mas consomem quantidades significativas de energia. Ainda segundo [Kumar & Lu, 2010], existem quatro abordagens básicas para economizar energia em dispositivos móveis:

- Adotar uma nova tecnologia: Com o avanço da tecnologia, transistores ficam menores e cada um consome menos energia mas mais transistores são necessários para a adição de novas funcionalidades e para a melhora do desempenho, voltando a aumentar o consumo energético.

- Evitar o desperdício de energia: Sistemas inteiros ou componentes individuais podem entrar em modo de espera para economizar energia.

- Executar programas mais lentamente: Quando a velocidade do clock de um processador dobra, o consumo energético chega ser quase 8 vezes maior [Kumar & Lu, 2010]. Se a velocidade do clock for dividida pela metade, o tempo de execução dobra, mas apenas um quarto da energia é consumida.

- Eliminar a computação por parte ou por completo: O dispositivo móvel não é o único responsável por executar a computação; ao invés disso, a computação é realizada em outro lugar, aumentando o tempo de vida da bateria.

O *offloading* de processamento corresponde à última abordagem citada.

2.2 Trabalhos Relacionados

A maioria dos trabalhos encontrados na literatura sobre *offloading* de processamento possui foco exclusivo em *smartphones*, com o objetivo de aumentar a eficiência energética dos aparelhos, tendo em vista que a bateria é um dos principais gargalos dos dispositivos móveis. Dispositivos IoT podem ter capacidade energética e poder de processamento ainda menores do que *smartphones* atuais, o que faz do *offloading* uma técnica com potencial de aplicação muito grande nesses dispositivos. Nesta seção serão apresentados alguns trabalhos relacionados que proveem soluções em *offloading* para dispositivos móveis. Os trabalhos relacionados estão apresentados em ordem cronológica crescente, sendo o primeiro de 2010 e o último de 2019.

O trabalho de [Cuervo et al., 2010] propôs um *framework* baseado na tecnologia .NET chamado MAUI, no qual o foco é a economia de energia. O MAUI utiliza um *profiler* para medir o consumo energético de uma aplicação e realizar estimativas para comparar o gasto energético ao executar o código no próprio aparelho e o gasto energético para transmitir os dados e códigos para a execução remota. Os autores avaliaram seu *framework* utilizando três aplicações, revelando que *offloading* de computação não apenas economiza energia, mas pode acelerar a execução das aplicações. MAUI possui motor de decisão elementar que monitora ativamente as atividades do dispositivo, o que causa impacto no desempenho do sistema. Nossa solução não monitora consumo energético de forma ativa, tendo menos impacto na utilização de recursos.

O trabalho de [Kemp et al., 2010] propôs Cuckoo, um *framework* simples para *offloading* de computação para a plataforma Android. Ele suporta a execução de código local e remota, baseado apenas na disponibilidade do servidor, sempre realizando o *offloading* de um método quando o servidor remoto está disponível. O Cuckoo também implementa uma biblioteca para gerenciar a comunicação entre dispositivo e servidor remoto. Cuckoo não possui motor de decisão, sempre realizando o *offloading* para os servidores remotos. Nem sempre realizar o *offloading* irá economizar tempo ou energia do dispositivo, então ter um motor de decisão é fundamental para uma melhor utilização dos recursos do dispositivo.

O trabalho de [Chun et al., 2011] propôs o CloneCloud, que realiza uma partição do código da aplicação em um conjunto de pontos de execução, de forma automatizada, utilizando análise estática. Para realizar as partições, são utilizadas informações sobre a qualidade da rede, velocidade de CPU e consumo de energia. Esses pontos de execução são determinados de forma que as partições sejam executadas no ambiente de execução mais eficiente, seja no próprio dispositivo ou em um clone do dispositivo na nuvem. Essa técnica também leva em consideração se o que está sendo otimizado é o tempo

de execução da aplicação ou a quantidade de energia consumida pelo dispositivo. O *framework* decide em tempo de execução se uma *thread* da aplicação deve ser migrada para o servidor na nuvem, caso as predições apontem economia de consumo energético ou no tempo de execução da aplicação. O motor de decisão do CloneCloud é feito de forma offline, não levando em consideração informações obtidas em tempo de execução. As informações utilizadas pelo motor do MLOOF podem melhorar a qualidade das predições.

O trabalho de [Verbelen et al., 2012] propôs AIOLOS, que utiliza informação da qualidade da rede e dos recursos do servidor para decidir, em tempo de execução, se um método deve ser executado localmente ou remotamente. Além disso, para cada chamada de método, o *framework* estima o tempo de execução local e remoto baseado no tamanho dos argumentos para realizar a decisão de rodar localmente ou remotamente. Após a execução do método, os dados mensurados são usados para atualizar os parâmetros de predição, otimizando a execução da aplicação levando em consideração as capacidades do dispositivo e a qualidade da conexão de rede. O motor de decisão do AIOLOS é similar ao motor do MLOOF, porém ele é um *framework* com muito mais intrusividade, fazendo com que o programador precise modificar partes do código para utilizá-lo.

O artigo de [Kosta et al., 2012] propôs ThinkAir, que possui um controlador de execução para decidir se a execução de um método deve ser realizada remotamente, baseado no tempo de execução, consumo de energia e no custo para utilizar a infraestrutura de um servidor na nuvem. O tempo de execução e consumo energético são modelados utilizando informação de execuções passadas. Sua *API* permite a especificação de quais métodos podem ser executados remotamente e o *framework* decide se a execução será local ou remota para cada chamada desses métodos. Um gerenciador de conexões foi desenvolvido para receber e executar o código remoto, bem como para retornar os resultados. ThinkAir possui um motor de decisão online, mas mais simplificado. O motor de decisão do MLOOF não realiza medições de consumo de energia ativamente em tempo de execução, tornando-o mais eficiente em questão de utilização de recursos.

O trabalho de [Shi et al., 2014] propôs COSMOS, um *framework* que utiliza uma estratégia gulosa para a decisão de *offloading*. Sempre que uma tarefa que pode ser executada remotamente é iniciada, o controlador determina se a execução remota será benéfica ou não. O controlador utiliza o tamanho dos argumentos dos métodos, largura de banda de *upload*, tamanho do resultado e largura de banda de *download* na decisão. As predições são atualizadas ao final de cada execução, ajustando as larguras de banda de *upload* e de *download* previstas. COSMOS possui alta intrusividade no desenvolvi-

mento da aplicação, fazendo com que o desenvolvedor precise moldar sua aplicação em volta do *framework*. Nossa solução possui baixa intrusividade, o desenvolvedor precisa apenas realizar poucas adições no código próximas aos métodos alvo e executar um pós-compilador e um novo arquivo compilado é gerado automaticamente.

O trabalho de [Flores et al., 2017] propôs HyMobi, um sistema de *offloading* híbrido que utiliza servidores em nuvem, *cloudlets* e comunicação D2D (*Device-to-Device*) para economizar energia de *smartphones* Android. No HyMobi os dispositivos são organizados em uma "comunidade" na qual a realização de computação para outros dispositivos gera créditos. Os aparelhos podem consumir créditos para executar métodos de suas aplicações em outros dispositivos. Não são feitas predições de consumo energético ou de tempo de execução dos métodos. As decisões de *offloading* são tomadas com base na pontuação de cada dispositivo participante da rede. HyMobi é um arcabouço que utiliza *cloudlets*, mas funciona de forma diferente ao MLOOF. As tomadas de decisão necessitam uma rede de dispositivos, enquanto que o MLOOF funciona de forma isolada, em um único dispositivo.

O trabalho de [Chen & Hao, 2018] propôs formulações para o problema de distribuir tarefas de dispositivos móveis para a borda da rede utilizando programação linear. A modelagem leva em consideração o tempo que a execução da tarefa duraria se executada no próprio dispositivo e o tempo que duraria se o *offloading* fosse realizado, além de utilizar a capacidade da bateria dos dispositivos como limitador de quanto processamento o dispositivo pode realizar.

Os autores de [Neto et al., 2018] propuseram ULOOF, um *framework* para *offloading* de processamento de *smartphones* para a nuvem, criado para ser transparente e não intrusivo, funcionar de forma simples, sem a necessidade de realizar constantes leituras do nível de energia do dispositivo, e levar a localização do usuário em consideração no seu processo de decisão. O programador que utiliza o ULOOF deve apenas importar a biblioteca do *framework* e utilizar marcações em sua aplicação, anotando quais são os métodos candidatos ao *offloading*. Após a compilação da aplicação, um pós-compilador modifica os métodos anotados, integrando a lógica do processo de *offloading* à aplicação. Este trabalho será discutido mais detalhadamente a seguir.

O trabalho de [Cheng et al., 2019] propôs uma arquitetura de *offloading* de computação de dispositivos IoT que utiliza a nuvem e *edge computing*. Veículos aéreos autônomos são usados como servidores de borda e a comunicação com os servidores na nuvem é feita via satélite. Um processo de decisão de Markov é utilizado na tomada de decisão de *offloading* utilizando em consideração as dinâmicas da rede juntamente com um algoritmo de aprendizado de máquina. Essa arquitetura utiliza modelos de decisão diferentes dos utilizados no MLOOF (utilizando algoritmos de aprendizado de

máquina). Ambos os modelos possuem suas vantagens e desvantagens, sendo que os modelos de aprendizado de máquina estão sendo mais utilizados em trabalhos recentes, porém são mais complexos e complicados de configurar corretamente.

2.2.1 Outros Trabalhos

Um ponto importante deste trabalho foi a adaptação de código existente para rodar em um dispositivo IoT específico. O trabalho de [Milinković et al., 2015] cita de forma geral algumas dificuldades existentes ao portar para dispositivos IoT código feito para outras plataformas. O trabalho de [Persson & Angelsmark, 2015] também discute dificuldades na portabilidade de código, e propõe um arcabouço que permita a execução de um mesmo código em dispositivos distintos.

Outro ponto importante do trabalho é sua arquitetura de três níveis, incorporando a utilização de *cloudlets* em sua hierarquia. O trabalho de [Shaukat et al., 2016] cita dificuldades gerais sobre a implantação e utilização de *cloudlets* em sistemas de *offloading*.

2.3 ULOOF

O ULOOF (*User-level Online Offloading Framework*) [Neto, 2016] é um *framework* para *offloading* de processamento de *smartphones* para a nuvem, e está utilizado como base para o trabalho apresentado nesta dissertação. O *framework* foi criado para ser transparente e não intrusivo, funcionar de forma simples, sem a necessidade de realizar constantes leituras do nível de energia do dispositivo, e levar a localização do usuário em consideração no seu processo de decisão.

O ULOOF foi desenvolvido para executar no sistema operacional Android e escrito na linguagem de programação Java, que é a linguagem padrão do Android, já que o foco era um *framework* que rodasse em nível de usuário. O programador que utiliza o ULOOF deve apenas importar a biblioteca do *framework*, que contém métodos e interfaces para monitoramento, instrumentação e comunicação com o servidor remoto, e deve utilizar marcações em sua aplicação, indicando quais são os métodos candidatos ao *offloading*. Essa abordagem simplifica o desenvolvimento, já que o programador não tem que se preocupar em como a biblioteca funciona internamente, ele apenas desenvolve sua aplicação normalmente e marca os métodos que podem ser executados na plataforma remota.

Junto ao *framework* foi desenvolvido um pós-compilador, que substitui o código original da aplicação, já compilada, pelo código do *framework* de *offloading*. Os méto-

dos marcados pelo programador como candidatos ao *offloading* são substituídos pelos métodos do ULOOF, que realizam a instrumentação do sistema - calculam tempo de execução e energia consumida pela execução dos métodos - e executam o *offloading* do código dependendo do resultado gerado pelo motor de decisão.

Caso a decisão escolhida para uma execução de um método seja a execução remota, a biblioteca envia ao servidor todos os dados necessários para a execução do método; Isso inclui o nome do método a ser executado, seus argumentos e o próprio objeto que implementa o método. Para isso funcionar, também foi desenvolvido junto ao ULOOF a aplicação que roda no servidor, respondendo às conexões e executando as aplicações dos clientes. A aplicação do servidor foi desenvolvida utilizando a mesma plataforma do cliente, ou seja o servidor é uma máquina rodando o sistema operacional Android, o que permite que os códigos dos clientes sejam executados de forma natural, e assume-se que um código a ser executado já esteja presente na plataforma remota.

2.3.1 Descrição do Sistema

Para utilizar o ULOOF em sua aplicação, o programador precisa apenas importar a biblioteca do *framework*, marcar os métodos que possam ser executados na plataforma remota - utilizando a funcionalidade de anotações da linguagem Java - e executar um pós-compilador em sua aplicação. Esse pós-compilador irá substituir os métodos marcados como candidatos ao *offloading* pelos métodos do ULOOF, que realizam a instrumentação do sistema e executam o *offloading* do código dependendo do resultado gerado pelo motor de decisão.

A biblioteca de *offloading* é composta por cinco módulos, que são:

- *Bandwidth Manager*: Responsável pelo monitoramento de mudanças na rede e pelo armazenamento de um histórico de qualidade da rede, que é utilizado pelo motor de decisão.

- *Execution Manager*: Monitora a execução de métodos e seus tempos de execução, quantidade de bytes transferidos pela rede, tamanho do resultado e ciclos de CPU consumidos. Esses dados são armazenados em um histórico que é utilizado pelo motor de decisão.

- *Energy Module*: Provê estimativas para o consumo de energia da CPU e do rádio. É utilizado nos cálculos feitos pelo motor de decisão.

- *Decision Engine*: O motor de decisão é o "cérebro" do *framework*. É ele quem decide se a execução de um método será feita localmente ou em um servidor

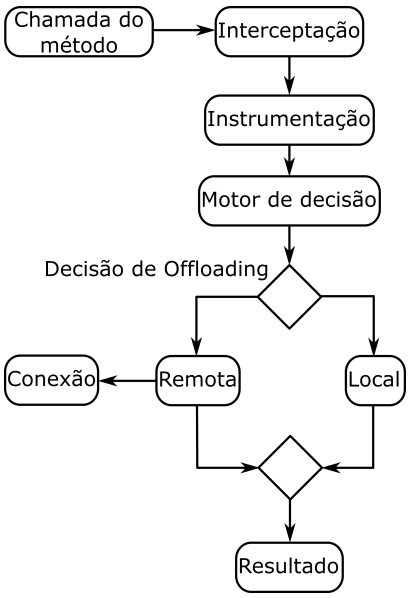

Figura 2.1. Diagrama de funcionamento da plataforma ULOOF.

remoto. A decisão é tomada de acordo com os parâmetros coletados pelos outros módulos - como tempos de execução e consumo de energia dos métodos. Se a qualidade da rede estiver muito ruim, a energia consumida para transmitir os dados de execução do método pela internet pode ser maior do que a energia consumida na execução local. Caso o método realize computação muito intensa, a transferência da execução para a plataforma remota pode economizar energia e até mesmo tempo. Os históricos armazenados pelos módulos *Execution Manager* e *Bandwidth Manager* e as estimativas presentes no *Energy Module* são utilizados para a previsão do tempo de execução e consumo energético de um método, que permitem a tomada de decisão.

- *Offloading Manager*: Instancia e inicializa os outros quatro módulos.

A Figura 2.1 ilustra o funcionamento da plataforma ULOOF. Quando um método da aplicação é executado, sua invocação é interceptada pelo *framework*, que realiza as operações de instrumentação e executa o motor de decisão. Se a decisão for de execução local, o método é executado normalmente no dispositivo. Se a decisão for para a execução remota, o código de conexão com a plataforma na nuvem é executado, dando início ao processo de *offloading*. Ao fim do processamento do método, o resultado é enviado de volta ao dispositivo e a execução termina.

2.3.2 Motor de Decisão

O motor de decisão é o módulo do *framework* responsável por executar cálculos de previsão e por decidir quando executar um método localmente e quando executá-lo remotamente. O motor de decisão utilizado no MLOOF é o mesmo motor desenvolvido para o ULOOF. Houve uma tentativa de melhoria no motor de decisão, porém os resultados não foram satisfatórios e as melhorias foram descartadas. Os outros módulos do arcabouço fazem a instrumentação do sistema, armazenam os dados de execuções passadas em um histórico de execuções e realizam as predições de futuras execuções utilizando este histórico. O gerenciador de energia utiliza equações de consumo baseadas em utilização de CPU e rádio do dispositivo. Estas equações são calculadas para o dispositivo específico e permitem que as predições sejam realizadas sem a necessidade de medir ativamente o consumo energético do dispositivo em tempo de execução. O gerenciador de rede armazena em seu histórico informação de banda e qualidade do sinal 4g. O gerenciador de execução utiliza os dados de execução (argumentos de entrada dos métodos) em tempo de execução e tamanho final do resultado. Para cada execução de um método, os argumentos, tempo de execução e tamanho do resultado são armazenados em um histórico de execuções. As predições de execuções desconhecidas são feitas interpolando os dados presentes no histórico. Após a execução do método, os valores reais de execução são guardados e melhoram a qualidade das próximas predições.

O motor de decisão trabalha com funções de utilidade de energia e tempo, ponderadas por uma constante α. A constante $0 \leq \alpha \leq 1$ representa uma inclinação para economia de energia ou para diminuição do tempo de execução. Quanto mais perto de zero, maior a inclinação para economizar energia, e quanto mais próxima de um, maior a inclinação para aumentar a responsividade da aplicação. As equações são apresentadas a seguir:

$$L(M) = \alpha * tl(M) + (1 - \alpha) * el(M)$$

$$R(M) = \alpha * tr(M) + (1 - \alpha) * er(M)$$

Nas equações, M indica o método que está sendo avaliado, $L(M)$ é a utilidade da execução local e $R(M)$ é a utilidade da execução remota. As funções $tl(M)$ e $tr(M)$ representam as estimativas de tempo de execução local e remota, respectivamente, e as funções $el(M)$ e $er(M)$ indicam as estimativas de consumo energético na execução local e remota, respectivamente. Dessa forma, a decisão escolhe o método de execução de acordo com a função de utilidade de menor custo:

Tabela 2.1. Tabela de comparação de recursos dos trabalhos.

Framework	**Motor de Decisão**	**Intrusividade**	***Cloudlets***	**IoT**
Cuckoo	Não	Alta	Não	Não
HyMobi	Não	Média	Sim	Não
MAUI	Sim, elementar	Baixa	Não	Não
ThinkAir	Sim, elementar	Média	Não	Não
CloneCloud	Sim, offline	Média	Não	Não
AIOLOS	Sim	Alta	Não	Não
COSMOS	Sim	Alta	Não	Não
ULOOF	Sim	Baixa	Não	Não
MLOOF	Sim	Baixa	Sim	Sim

$$Offloading(M) = R(M) < L(M)$$

2.4 Conclusão

A Tabela 2.1 apresenta uma comparação entre os recursos presentes nos trabalhos relacionados, incluindo o ULOOF. Os recursos comparados são: (a) Intrusividade, indicando como a utilização do *framework* afeta o desenvolvimento/execução da aplicação, seja por modificações em tempo de execução, modificação em código ou modificação na forma em que a aplicação é desenvolvida; (b) Motor de Decisão, indicando se a solução possui motor de decisão preciso; (c) Modelo de Energia, indicando se um modelo de consumo energético foi utilizado e como ele é atualizado, de forma offline ou online; (d) Nível de Usuário, indicando se o *framework* foi implementado em nível de usuário do sistema operacional; e (e) Plug-and-play, indicando se o *framework* pode ser facilmente instalado e utilizado.

Esse trabalho utiliza o ULOOF como base, aplicando melhorias no que já foi feito e implementando novos algoritmos no sistema.

Capítulo 3

MLOOF - Multi-level Online Offload Framework

Neste capítulo descrevemos o MLOOF, nossa solução de *offloading* para a internet das coisas e para *smartphones*. O MLOOF é organizado em uma hierarquia de três níveis (dispositivos, *cloudltes* e servidores na nuvem), trazendo mais flexibilidade ao processo de *offloading* junto a um aumento na vazão da rede e diminuição de latência, na qual os clientes podem ser dispositivos tipo *smartphones* ou IoT.

O sistema proposto é conceitualmente simples, organizado em uma hierarquia de três níveis. As três entidades principais são os dispositivos clientes, os servidores na *cloudlet* e os servidores na nuvem, todas com capacidade de processamento e comunicação. Os dispositivos clientes executam originalmente as aplicações dos usuários. Os servidores na nuvem e na *cloudlet* têm, por definição, capacidade de executar as mesmas aplicações que executam no dispositivo, mas o fazem apenas quando solicitados. Para a comunicação entre essas entidades, considera-se que existe uma infraestrutura de rede, por meio da qual os dispositivos se conectam à *cloudlet* com baixa latência e possivelmente altas taxas de dados. De maneira semelhante, os dispositivos podem se conectar diretamente à nuvem para fazer o *offloading*, normalmente com qualidade de conexão pior quando comparada à conexões com *cloudlets*, visto que a nuvem pode estar situada em qualquer lugar, o que pode levar a altos atrasos de comunicação.

A Figura 3.1 mostra uma esquematização do sistema proposto. Os clientes se conectam a uma *cloutlet*, se houver alguma disponível, ou diretamente ao servidor na nuvem. As *cloudlets* permitem uma conexão com baixa latência, essencial para aplicações com requisitos de tempo real. O processo de decisão, que define se a execução de um método será realizado local ou remotamente, é realizado pelo dispositivo. A execução do método ocorrerá na *cloudlet* de preferência, por razões de eficiência, ou na

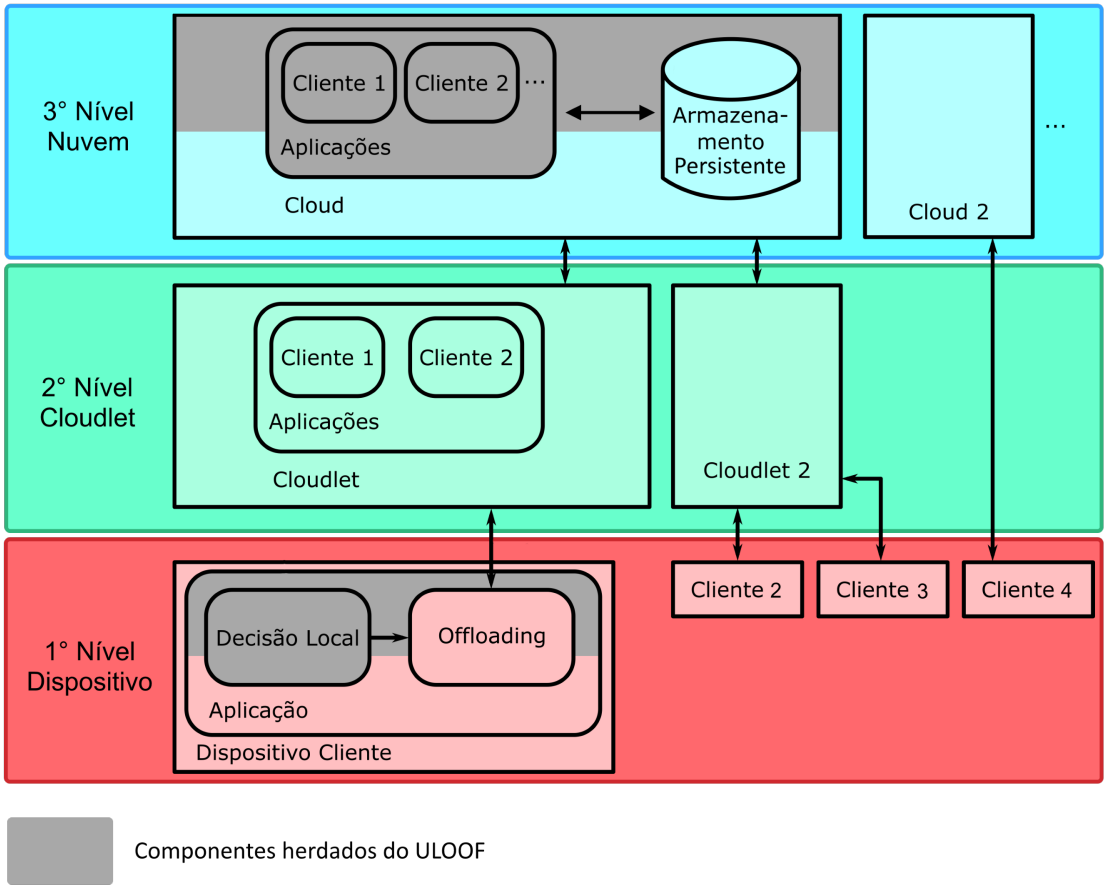

Figura 3.1. Arquitetura do sistema em três níveis.

nuvem, caso a *cloudlet* esteja sobrecarregada ou o cliente esteja conectado diretamente à *cloud*.

Cada dispositivo é responsável por estimar a demanda de uma certa computação, antes da sua execução, considerando o tempo de processamento e da transmissão de dados necessária em caso de execução remota, bem como o consumo de energia. Há um processo de decisão que define se a execução de um método (ou a computação) será executado local ou remotamente. Essa decisão é tomada pelo dispositivo, utilizando informações de execuções passadas para estimar e comparar os custos envolvidos nas execuções local e remota. Havendo decisão de execução remota do método, ela ocorrerá preferencialmente na *cloudlet*, por razões de eficiência, mas pode ser transferida para a nuvem, caso o servidor na *cloudlet* esteja sobrecarregado.

Este trabalho é uma extensão do ULOOF (*User-level Online Offloading Framework*) [Neto et al., 2018]. O *framework* foi ampliado para conter mais um nível hierárquico (de *cloudlet*). Isso implica modificações no protocolo e nas mensagens de comunicação tanto entre dispositivos e servidores quanto entre os próprios servidores,

que devem ser coordenados para servir ao mesmo cliente. Os dispositivos devem ser capazes de identificar e conectar-se ao servidores *cloudlet* em sua rede local automaticamente ou a servidores na nuvem caso as *cloudlet* não sejam identificadas. Duas principais características do ULOOF foram mantidas no MLOOF, sendo elas o baixo impacto do arcabouço em tempo de execução e a baixa intrusividade no desenvolvimento. O objetivo principal do MLOOF é evitar o desperdício de recursos do dispositivo, sendo assim o baixo impacto em desempenho é fundamental. Chamamos de intrusividade o impacto que a adoção do arcabouço traz para o desenvolvedor. Alta intrusividade significa que passar a utilizar o arcabouço obriga o desenvolvedor a realizar várias mudanças no código, incluindo mudanças na engenharia do software (por exemplo, forçar o desenvolvedor a chamar uma função de tomada de decisão sempre que um método candidato ao *offloading* for executado).

Foram também desenvolvidos os servidores que rodam nas *cloudlets*. Esses servidores tem como características principais a proximidade física aos dispositivos clientes e facilidade de instalação. Toda a configuração necessária para fazer com que um novo servidor em uma *cloudlet* funcione de forma transparente ao dispositivo cliente (como se fosse um servidor em nuvem previamente configurado) é feita pela *cloudlet* de forma automática.

O arcabouço foi todo reprojetado para rodar em dispositivos IoT, sem perder a capacidade de rodar em smartphones Android. Todo o código sofreu alterações para garantir compatibilidade com os dispositivos que não rodam o sistema operacional Android. Código específico (como partes do código de instrumentação) foi criado para executar em um Raspberry Pi, dispositivo utilizado durante experimentos rodando em hardware.

Essas novas características implementadas possibilitam a utilização do arcabouço em aplicações que possuam fortes restrições no tempo de execução, que seriam inviáveis de serem executadas na presença da alta latência existente na comunicação com servidores fisicamente distantes dos dispositivos.

As próximas seções explicam com maiores detalhes o papel de cada componente do sistema, denominado MLOOF (*Multi-level Online Offload Framework*).

3.1 Primeiro Nível - Dispositivo

Um dispositivo móvel (*smartphone* ou dispositivo IoT) pode executar código de uma aplicação utilizando seus próprios recursos (execução local) ou enviar uma requisição pela internet para que esse código seja executado em uma outra máquina (execução

remota). Optando-se pela execução local, o dispositivo utiliza seus próprios recursos computacionais para rodar a aplicação, o que demora uma certa quantidade de tempo e consome certa energia do dispositivo. Optando-se pela execução remota, o dispositivo envia pela rede a informação de qual código deve ser executado e quais dados devem ser utilizados, e aguarda o resultado. Na execução remota, gasta-se tempo para realizar a comunicação entre dispositivo e servidor e para a computação no servidor, bem como consome-se energia para enviar e receber os dados pela rede. O tempo gasto e a energia consumida em ambos os casos dependem de diversas fatores, como a qualidade da rede utilizada, o consumo energético do processador e do rádio dos dispositivos, o poder de processamento do dispositivo e do servidor e a distância entre eles.

Utilizando informações coletadas sobre execuções passadas no dispositivo, é possível realizar predições de tempo de execução e consumo energético de futuras execuções. Essas predições são importantes para que a decisão sobre onde o código deve ser executado possa ser tomada. Caso a estimativa de tempo de execução local seja maior do que o tempo da execução remota, a decisão de realizar o *offloading* leva à economia de tempo. Caso a estimativa de consumo energético para enviar/receber os dados pela rede seja menor do que a estimativa de consumo para executar o código localmente, a não realização de *offloading* deve economizar energia. O funcionamento do motor de decisão é explicado em mais detalhes na Seção 2.3.2.

3.1.1 Instrumentação

O *framework* faz predições de tempo de execução e consumo energético de futuras execuções de métodos utilizando um histórico de execuções passadas. Duas execuções de um mesmo método com argumentos similares devem consumir quantidades similares de recursos. Caso o método não se comporte de forma previsível, as primeiras execuções poderão ter predições erradas, mas o histórico será atualizado com os valores das próximas execuções e as próximas predições serão mais precisas.

Um dos requisitos importantes deste *framework* de *offloading* é que deve haver baixo impacto no desempenho. Para que esse requisito seja cumprido, não são feitas medições de consumo energético durante a execução da aplicação; São feitas de forma *offline*, gerando fórmulas de consumo energético por utilização do processador e por transmissão de dados pela rede, que são codificadas no *framework* de forma estática. Durante a execução da aplicação, são monitorados o consumo de CPU e a quantidade de dados enviados e recebidos pela rede. Esses valores são usados para estimar o consumo imediato de energia em tempo de execução.

O consumo de CPU é medido utilizando a métrica de *CPU ticks* provido pelo

sistema operacional Linux, correspondente à carga exercida no processador. O processador do dispositivo é um recurso compartilhado entre vários processos rodando no sistema, o que faz de *CPU ticks* uma ótima unidade de medida para calcular a taxa de utilização do processador por um método específico, já que o sistema possui um contador de *ticks* separado para cada processo. *CPU ticks* é uma métrica normalizada em cem *ticks* por segundo. Para calcular a taxa de utilização do processador consumida por um método, basta calcular a diferença de *ticks* depois e antes de executar o método e dividir pelo tempo gasto.

A energia gasta para transmitir informações pela rede é medida de acordo com a quantidade de bytes transmitidos. Em tempo de execução, a quantidade de bytes transmitidos é utilizada para estimar o consumo energético causado pela utilização da rede.

3.1.2 Offloading

O arcabouço de *offloading* é composto por quatro módulos principais: (1) O Gerenciador de Rede, que armazena as informações e realiza predições sobre a qualidade da rede; (2) O Gerenciador de Execução, que armazena o tempo de execução dos métodos e realiza predições de execuções futuras; (3) O Módulo de Energia, que estima o consumo de energia do processador e da utilização da rede; (4) O Motor de Decisão, que utiliza os outros três módulos para optar pela execução remota ou local dos métodos. A Figura 3.2 apresenta a relação entre os quatro módulos. Caso a decisão seja executar o *offloading*, o motor de decisão realiza o processo de comunicação com o servidor, envia os dados e aguarda a chegada da resposta. Caso a opção escolhida seja a de executar o método localmente, o motor de decisão faz a chamada do método, que executa e retorna o resultado obtido.

3.2 Segundo Nível - Cloudlet

O servidor da *cloudlet* é o recurso principal, ou alternativa preferencial, utilizado pelos dispositivos para a realização do *offloading*. Se o servidor recebe uma requisição para executar determinado método com determinados argumentos, ele executa o método e transmite o resultado de volta para o dispositivo. Caso não haja nenhuma *cloudlet* disponível ou a *cloudlet* não possua capacidade para atender à requisição, o dispositivo pode se conectar diretamente a um servidor na nuvem.

Ao receber um pedido de *offloading*, o servidor da *cloudlet* primeiramente se comunica com o servidor na nuvem e verifica se existe uma versão mais atualizada da

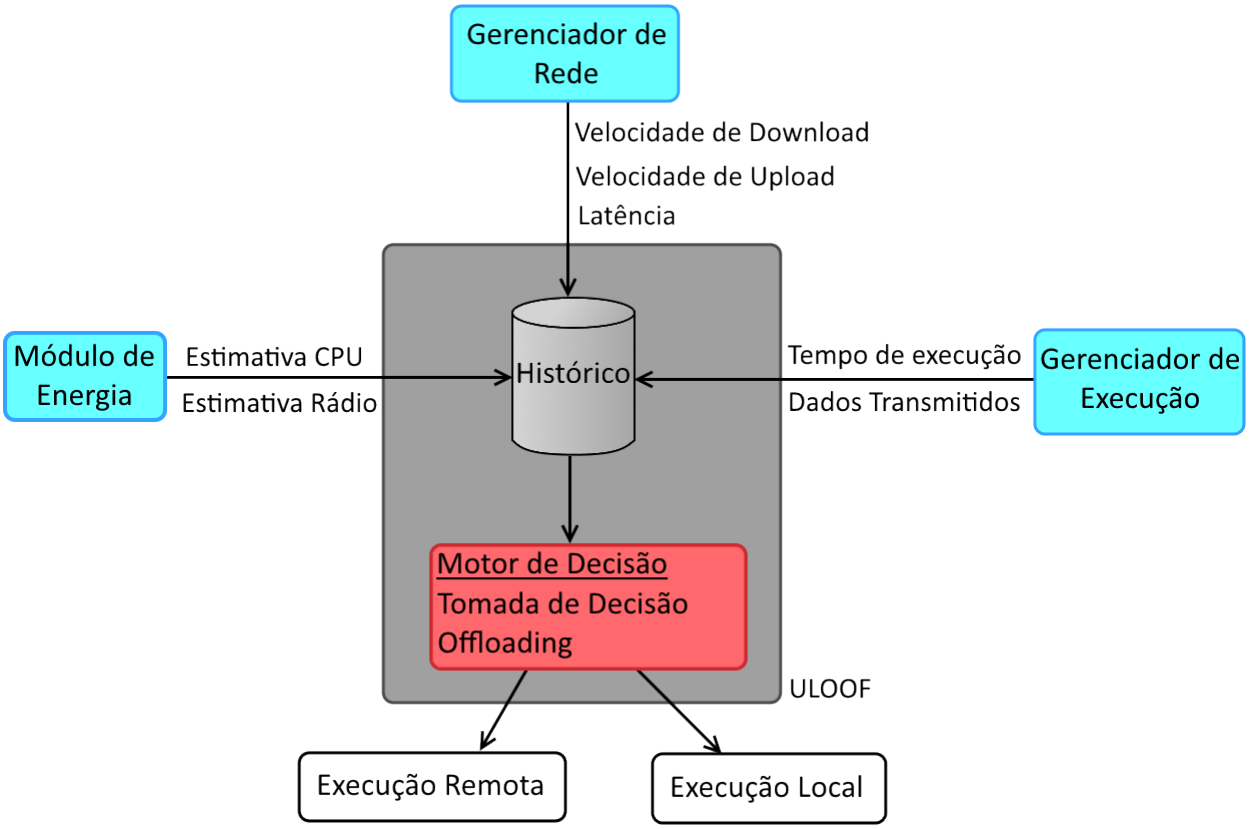

Figura 3.2. Diagrama de relação entre os módulos de *offloadding*.

aplicação. Caso a *cloudlet* não possua o código da aplicação ou exista uma versão mais atualizada, o servidor baixa o código de um servidor na nuvem e dá continuação ao processo de *offloading*. Uma das características e limitações do MLOOF é a necessidade dessa aplicação estar presente a priori em algum servidor na nuvem. Se por um lado essa solução adiciona uma etapa adicional de preparação antes do usuário poder utilizar o *framework*, na qual o código da aplicação deve ser enviado manualmente aos servidores na nuvem, por outro lado essa abordagem elimina custos adicionais de enviar o código pela rede em tempo de execução (a aplicação já está presente nos servidores antes dos dispositivos clientes tentarem executá-la), economizando tempo de processamento e carga de bateria. Esse código enviado aos servidores é o código binário já compilado, o que aumenta a segurança dos usuários por não terem que expor o código fonte de suas aplicações.

O servidor da *cloudlet* baixa do servidor da nuvem o código compilado da aplicação do cliente. Para poder executar os métodos solicitados durante o processo de *offloading*, o servidor precisa carregar esse código compilado na sua própria instância da máquina virtual do Java. Para conseguir fazer isso, o servidor da *cloudlet* utiliza

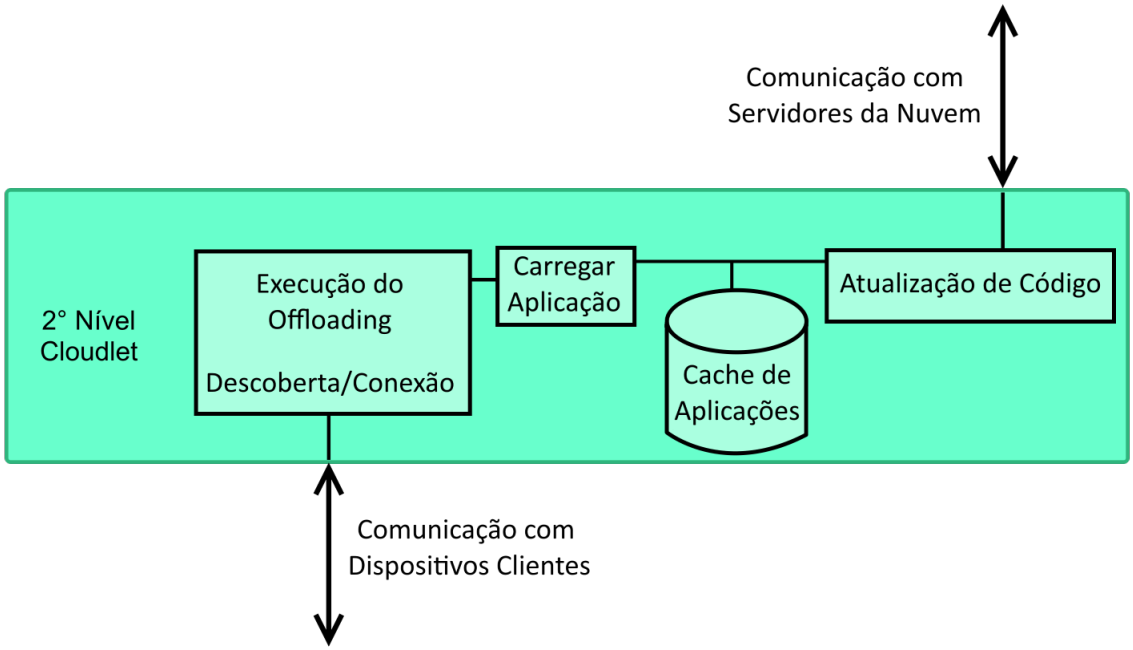

Figura 3.3. Diagrama de componentes presentes no servidor da cloudlet.

a API de reflexão da linguagem Java, o que permite a importação de código de um arquivo e acesso aos seus métodos e classes.

A Figura 3.3 apresenta um diagrama com os componentes presentes no servidor da *cloudlet*. Estes componentes podem ser assim descritos:

- **Descoberta de dispositivos e criação da conexão inicial**: Em nosso arcabouço de *offloading*, assumimos que o servidor da *cloudlet* possui duas características. A primeira é que ele deve estar localizado fisicamente próximo ao dispositivo, permitindo uma conexão com alta vazão e, principalmente, menor latência. A segunda característica é que o servidor da *cloudlet* possui fácil instalação, permitindo que um novo servidor seja criado rapidamente caso necessário. Para atingir esse objetivo, os dispositivos clientes realizam a descoberta automática de *cloudlets* na rede local (utilizando *broadcast* UDP) e iniciam o processo de conexão com o servidor. Uma descrição mais sucinta pode ser visualizada no Algoritmo 1.

- **Execução de *offloading***: Realiza a execução do *offloading* propriamente dito. Isso inclui a deserialização da informação enviada pelo dispositivo, a identificação do arquivo contendo o código requisitado, a atualização do código com algum servidor na nuvem (caso o código contido na cache da *cloudlet* esteja desatualizado), a execução do método requisitado e o envio do resultado ao dispositivo cliente.

- **Atualização de código**: Os códigos dos métodos requisitados pelos dispositivos clientes são armazenados em uma cache no servidor da *cloudlet* em forma de arquivos "Jar" compilados. Antes de executar o método requisitado pelo dispositivo, o servidor da *cloudlet* faz uma verificação de versão com algum servidor da nuvem. Caso o servidor da nuvem possua uma versão mais atualizada do que a existente na cache do servidor da *cloudlet*, essa nova versão é baixada pela *cloudlet* e a execução do código continua normalmente. Uma descrição mais sucinta pode ser visualizada no Algoritmo 2.

- **Carregamento da aplicação**: Para poder executar o código requisitado pelos dispositivos, o servidor da *cloudlet* primeiramente carrega o arquivo que contém o código compilado em sua própria máquina virtual Java. Com o arquivo carregado, o servidor consegue executar os métodos como se fosse um método compilado com o próprio servidor, utilizando técnicas de reflexão.

Algorithm 1: Algoritmo de Descoberta de Dispositivos

```
1 Function Descoberta():
2     BroadcastUDP(Id);
3     ListaDeCloudlets[] = RespostasAoBroadcast(Id);
4     foreach Cloudlet ∈ ListaDeCloudlets do
5         if TentativaDeConexão(Cloudlet) == Verdadeiro then
6             Retorna Cloudlet;
7         end
8     end
9 return
```

Algorithm 2: Algoritmo de Atualização de Código

```
1 Function AtualizaCodigo(C):
2     VersaoEmCache = VerificaVersaoCodigo(C);
3     AtualizaCodigoComServidorNuvem(C, VersaoEmCache);
4 return
```

Na versão atual do sistema, apenas uma versão de cada aplicativo é suportada. O servidor da *cloudlet* identifica a aplicação pela assinatura do método enviado pelo dispositivo cliente e faz a requisição da versão mais recente desta aplicação que o servidor na nuvem possuir. Se existissem vários dispositivos clientes que possuíssem versões diferentes da mesma aplicação, o sistema não iria funcionar corretamente pois apenas a versão mais recente seria suportada. A solução para esse problema é relativamente simples, precisamos apenas identificar a versão atual do aplicativo no dispositivo cli-

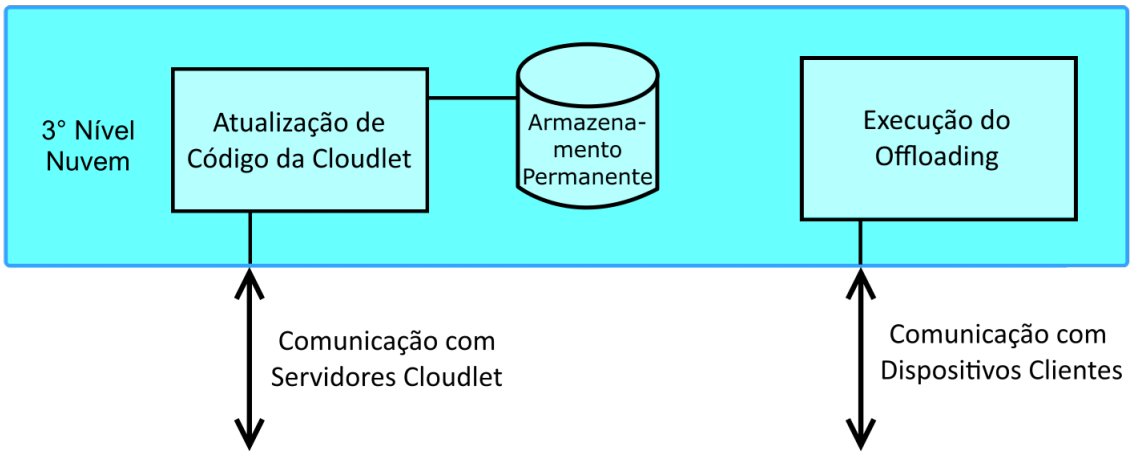

Figura 3.4. Diagrama de componentes presentes no servidor da nuvem.

ente e enviar a versão aos servidores de *offloading* para que a aplicação correta possa ser carregada. Optamos por deixar esse tipo de melhoria para trabalhos futuros.

3.3 Terceiro Nível - Nuvem

O servidor na nuvem serve como segunda alternativa para a realização do *offloading* por parte dos dispositivos. Ademais, tem a função de guardar os códigos das aplicações que rodam nos dispositivos, para que ele possa executá-los caso haja uma requisição de *offloading* por parte dos dispositivos, e para que ele possa enviar a aplicação para as *cloudlets* caso elas ainda não possuam o código ou possuam um código desatualizado.

O servidor da nuvem pode atender os dispositivos clientes da mesma que servidores das *cloudlets*, porém os dispositivos dão prioridade às *cloudlets* por estarem fisicamente próximas aos dispositivos (normalmente na mesma rede local), o que possibilita uma conexão de melhor qualidade (maior largura de banda e menor latência).

A Figura 3.4 apresenta um diagrama com os componentes presentes no servidor da nuvem. Estes componentes podem ser separados em:

- **Atualização de código da *cloudlet***: Um dos requisitos do MLOOF é que os códigos compilados das aplicações dos dispositivos clientes devem estar previamente instalados no servidor da nuvem. Isso permite que o servidor da nuvem execute esses códigos no caso de uma requisição de *offloading* e que possa enviar esses códigos para os servidores da *cloudlet*. Esses códigos são armazenados em uma cache no servidor da *cloudlet* em forma de arquivos "Jar" compilados. Ao receber um pedido de *offloading*, a *cloudlet* realiza a verificação de versão desses arquivos em cache com um servidor da nuvem conhecido. Caso o servidor da

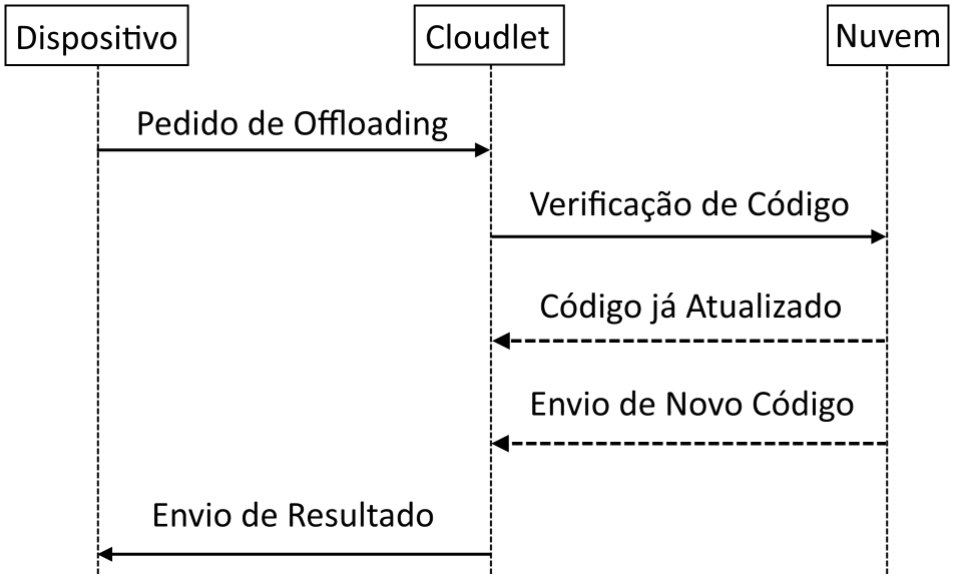

Figura 3.5. Diagrama de sequência de comunicação entre os níveis do sistema com atualização de código em tempo de *offloading*.

nuvem possua uma versão mais atualizada do código, ele envia esse novo código para o servidor da *cloudlet*, que pode continuar a servir os clientes com esse novo código.

- **Execução do *offloading***: Os servidores na nuvem podem servir aos clientes da mesma forma como os servidores das *cloudlets* fazem. Isso acontece quando não há servidor *cloudlet* na mesma rede local dos dispositivos, ou quando esses servidores estão sobrecarregados e não conseguem atender novos pedidos de *offloading*. Nesses casos, os dispositivos iniciam a conexão com um servidor da nuvem e o processo de *offloading* continua normalmente.

A Figura 3.5 representa a comunicação inicial entre dispositivo e servidores, quando o servidor da *cloudlet* ainda não possui o código a ser executado. Após o dispositivo realizar o pedido de *offloading*, a *cloudlet* verifica com o servidor na nuvem se existe um código mais atualizado. Se houver, o servidor na nuvem envia o código à *cloudlet*, que pode continuar o processo de *offloading* normalmente. Vale salientar que a comunicação entre o servudir da *cloudlet* e o servidor da nuvem acontece apenas uma vez durante a conexão inicial entre dispositivo e *cloudlet*. Ao baixar o código atualizado do servidor da nuvem, o servidor da *cloudlet* não precisa mais se comunicar com ela.

Tabela 3.1. Lista de mensagens para comunicação HTTP.

Identificador	Dados Enviados	Dados Respondidos
ping	Nenhum dado enviado	Nenhum dado respondido
execute	Dados serializados necessários para a execução de um método	Resultado serializado da execução do método
cloudlet	Identificador de requisição por parte da cloudlet	Resposta enviada pela cloud para a cloudlet

3.4 Comunicação Interníveis

Os três níveis do sistema (dispositivos, *cloudlets* e nuvem) se comunicam via rede utilizando um protocolo em comum. O protocolo, baseado em HTTP, determina como as mensagens devem ser organizadas pelo transmissor para que possam ser entendidas pelo receptor.

A comunicação utiliza URLs base na forma *http://IP:PORTA/* para identificar um servidor e utiliza o método HTTP GET para identificar as mensagens. Dessa forma, a comunicação de um dispositivo para um servidor é feita pelo URL *http://IP:PORTA/execute* no momento em que o dispositivo faz um pedido de *offloading*. Os parâmetros do método GET são usados para identificar o método a ser executado, enquanto que o corpo da requisição HTTP contém os dados serializados enviados pelo dispositivo, contendo todos os parâmetros necessários para que o servidor possa executar o método requisitado. Após o servidor receber a requisição ele pode executar o método, serializar a resposta e enviá-la novamente ao dispositivo como resposta à requisição HTTP inicial.

A Tabela 3.1 lista os principais tipos de mensagens utilizadas na comunicação com os servidores. Mensagens do tipo *ping* são utilizadas para medições de latência entre dispositivos e servidores e entre *cloudlets* e servidores da nuvem. Mensagens do tipo *execute* são utilizadas durante requisição de *offloading*. Os dados enviados nesse tipo de mensagem são a assinatura do método a ser executado e os parâmetros que o método necessita. Os parâmetros podem ser qualquer tipo suportado pela linguagem Java. Por isso, primeiramente, acontece a serialização desses parâmetros para que possam ser enviados ao servidor. Ao receber uma mensagem desse tipo, o servidor realiza a deserialização dos dados, obtendo os objetos originais, realiza a chamada da função, serializa o resultado e o envia para o cliente. As mensagens do tipo *cloudlet* são reservadas para comunicação entre servidor *cloudlet* e servidor da nuvem. Os dados enviados nessas mensagens identificam o tipo de requisição feita pela *cloudlet*. Essas mensagens podem ser: Requisitar a quantidade de arquivos executáveis presentes no servidor da nuvem; Requisitar que o servidor da nuvem envie um arquivo específico

para o servidor da *cloudlet*.

Uma das características dos servidores *cloudlet* do MLOOF é a fácil instalação, permitindo que um novo servidor seja criado rapidamente caso necessário. Para atingir esse objetivo, os dispositivos clientes realizam a descoberta automática de *cloudlets* na sua rede local. Para conseguir isso, um *broadcast UDP* é realizado, no qual pequenos pacotes UDP contendo uma mensagem identificadora são enviados para todas as máquinas na rede local do dispositivo. Cada servidor nessa rede local que receber a mensagem responde com uma outra mensagem para o endereço de origem, indicando que esse servidor está apto a atender os pedidos de *offloading* desse novo cliente. Os clientes armazenam uma lista com todos os servidores descobertos em sua rede mas, caso não haja nenhuma resposta, o cliente inicia a conexão e envia os pedidos de *offloading* para algum servidor conhecido na nuvem.

Toda a comunicação feita entre as entidades ativas utilizando o MLOOF é baseada nas URLs que identificam o tipo de mensagem suportada pelos servidores. Não é feita nenhuma confirmação sobre a identificação dos clientes, os dados enviados não são criptografados e os servidores não fazem nenhum tipo de verificação de segurança. Para a utilização do arcabouço em um ambiente de produção, essas medidas de segurança devem ser implementadas e podem ser adicionadas em trabalhos futuros.

3.5 Implementação

Os programas do arcabouço de *offloading*, servidor da *cloudlet* e servidor da nuvem foram implementados na linguagem de programação Java. A linguagem Java permite a execução de um mesmo código em variadas plataformas de hardware e diferentes sistemas operacionais, incluindo *smartphones* Android, computadores pessoais e outros hardwares rodando o sistema operacional Linux.

Aplicações desenvolvidas em Java podem ser facilmente modificadas para utilizar o arcabouço de *offloading*. O desenvolvedor escolhe métodos candidatos ao *offloading* e os anota com a marcação *@OffloadCandidate*. O código já anotado e compilado é então modificado por um pós-compilador, que modifica os métodos anotados para adicionar a lógica de *offloading*, que inclui código para realizar a instrumentação do dispositivo em tempo de execução, criação do histórico de execução dos métodos, motor de decisão e do processo de *offloading* propriamente dito. A Figura 3.6 ilustra as etapas necessárias para a geração da aplicação, desde a escrita do código original até a geração da aplicação final. A única modificação que o usuário deve fazer ao código original é adicionar a marcação *@OffloadCandidate* aos métodos candidatos ao *offloading*. Após isso o código

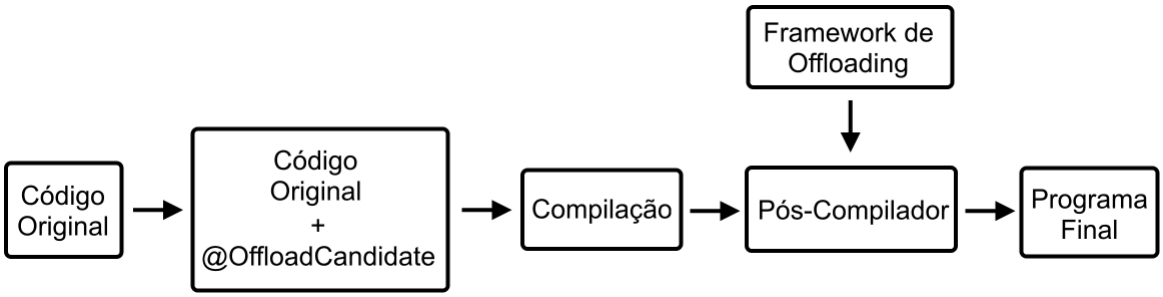

Figura 3.6. Diagrama de processo de geração da aplicação final.

é compilado normalmente e passa por um processo extra de pós-compilação, na qual toda a lógica do arcabouço de *offloading* é implantada no arquivo binário, gerando a aplicação final.

Os aplicativos utilizados para avaliação do sistema também foram desenvolvidos em Java e são explicados com mais detalhes na próxima seção.

Diversos fatores fizeram com que escolhêssemos a linguagem de programação Java para o desenvolvimento de todos os componentes do MLOOF. Primeiramente, Java foi a linguagem utilizada pelo ULOOF, trabalho no qual o MLOOF se baseia, por ser a linguagem mais utilizada no desenvolvimento de aplicações para o sistema operacional Android. Trocar de linguagem implicaria na necessidade de reprogramar todos os componentes do ULOOF, que estão presentes no MLOOF, nessa nova linguagem. O MLOOF foi projetado para ser utilizado por diversos dispositivos distintos e Java é uma linguagem bastante portável, permitindo sua utilização em diversos hardwares e sistemas operacionais sem a necessidade de muita alteração no código fonte das aplicações. Outra linguagem de programação (como C, por exemplo) poderia ser utilizada para desenvolver o MLOOF, mas traria grandes dificuldades para a implementação e utilização do código em dispositivos diversos, além de diminuir a característica de baixa intrusividade do sistema.

3.6 Motor de Decisão Distribuído

O motor de decisão do MLOOF armazena informações de execuções passadas e executa algoritmos utilizando estes dados para realizar predições de novas execuções. O motor de decisão é uma parte fundamental de todo o arcabouço pois boas predições farão com que o melhor ambiente de execução seja escolhido, o que trará maiores ganhos em desempenho e economia de energia. O próprio motor de decisão consome memória e tempo de CPU do dispositivo para poder armazenar e processar os dados

de instrumentação.

Com o objetivo de melhorar a qualidade das predições foi desenvolvido um algoritmo de tomada de decisão distribuída, onde os dados de execução de vários dispositivos são enviados aos servidores na nuvem, que criam regras básicas para auxiliar na tomada de decisão dos dispositivos clientes.

Os dispositivos enviam periodicamente aos servidores os dados de instrumentação armazenados em seus históricos de execução. Os servidores utilizam esses dados para gerar regras simples, que possam ser enviadas aos outros dispositivos sem gerar muito *overhead* na comunicação. As regras geradas utilizam como base os argumentos de entrada para o método, velocidades de download e upload, e latência da rede, além da quantidade de tempo e energia foram gastos para o dispositivo executar tal método. Em posse desses dados o servidor consegue criar limites nos quais possivelmente é mais vantajoso executar o método local ou remotamente. Por exemplo, para um método específico o servidor pode decidir que a partir de certos valores de entrada e quando a qualidade da rede não estiver boa (velocidades de download e upload abaixo de certo valor e latência acima de certo valor) é mais vantajoso para o dispositivo executar o método localmente para que o tempo de execução seja o menor possível.

Este algoritmo não obteve muito sucesso, adicionando *overhead* e não conseguindo gerar boas regras de predição. Existem alguns problemas fundamentais com a abordagem escolhida. Primeiramente, as regras geradas não tem o objetivo de substituir o motor de decisão presente nos dispositivos, pois elas nunca conseguiriam ser tão precisas quanto o motor que tem acesso direto aos dados de instrumentação do dispositivo em questão. Além disso, as regras funcionariam apenas para dispositivos com características de poder de processamento e consumo energético similares. Mesmo com essas características presentes nos testes realizados, a tomada de decisão distribuída não se mostrou vantajosa pois não conseguia gerar boas predições e fazia com que os dispositivos optassem por um dos ambientes de execução em detrimento ao outro (execução local ou remota), o que tornava a obtenção dos dados de instrumentação mais lenta, por parte dos dispositivos. Enquanto as condições para a execução remota não eram favoráveis, o dispositivo teria a tendência a executar seus métodos localmente, o que faz com que ele tenha menos dados de execuções remotas. Dessa forma o motor de decisão local ao dispositivo, que é mais preciso do que o motor de decisão distribuído, possui dados menos heterogêneos a sua disposição, demorando mais para conseguir gerar predições mais precisas. Por isso optamos por descontinuar a ideia de um motor de decisão distribuído e utilizar outras técnicas para a melhoria do motor de decisão em trabalhos futuros.

3.7 Conclusão

Este capítulo descreveu em detalhes a arquitetura e os componentes do MLOOF, um arcabouço de *offloading* de processamento dividido em três camadas. Seu desenvolvimento consistiu na implementação de distintos componentes, estando presente nos dispositivos clientes e nos servidores das *cloudlets* e da nuvem, sempre conservando os objetivos de possuir baixa intrusividade no desenvolvimento das aplicações e baixo impacto em desempenho durante a execução. Nos capítulos a seguir iremos realizar testes em ambiente real e simulações para avaliar o desempenho da nossa solução.

Capítulo 4

Avaliação, Experimentos e Análise

O objetivo da avaliação é verificar se e quanto a solução proposta atende aos propósitos de liberação de carga e melhoria no tempo de execução de determinadas tarefas de um dispositivo. Para isso, foram realizados experimentos em ambiente real e experimentos simulados com o objetivo de avaliar o impacto que a nossa estratégia de *offloading* pode causar na execução da aplicação.

Para um dispositivo que executa uma aplicação, um arcabouço de *offloading* pode ser útil para: economizar energia consumida pelo dispositivo; aumentar de forma indireta o poder de processamento do dispositivo; diminuir o tempo de execução de tarefas computacionais da aplicação. Dessa forma, as **métricas** a serem avaliadas pelos experimentos são:

1. **Tempo de execução** do método, definido como o tempo transcorrido entre a tomada de decisão pelo motor de decisão e obtenção do resultado no dispositivo. No caso de execução remota, inclui o tempo de transmissão dos parâmetros, execução no servidor remoto (*cloudlet* ou nuvem) e de transmissão dos resultados para o dispositivo. No caso de execução local, é apenas o tempo de execução do método no dispositivo;

2. **Consumo de energia** pelo dispositivo, definido como o gasto energético para a execução do método em questão, definido como, para execução local, o consumo desde o início e até o final da execução da tarefa e, para execução remota, o consumo de energia para transmissão de dados (parâmetros de entrada) e de recepção dos resultados.

O processo de avaliação do MLOOF foi dividido em três experimentos, cada um com seus objetivos. Os experimentos são:

1. **Experimento em Ambiente Controlado. Objetivo: verificação da funcionalidade e desempenho:** O primeiro experimento serviu como base para a avaliação funcional do arcabouço, indicando o momento quando ele estaria pronto para passar por testes mais completos, além de servir como uma avaliação inicial de desempenho. Os testes foram feitos com um dispositivo IoT, um servidor *cloudlet* e um servidor na nuvem. Como dispositivo IoT foi utilizado um Raspberry PI 3 e como servidores na *cloudlet* e *cloud* foram utilizados computadores pessoais. Os experimentos foram realizados em condições bem controladas, sendo que os servidores estavam sendo utilizados exclusivamente para os testes e a rede local não estava sendo compartilhada.

2. **Experimento em um Testbed. Objetivo: verificação de influências externas:** No segundo experimento foram utilizados a infraestrutura de um *testbed* para servir como servidor *cloudlet*, um servidor em nuvem comercial e o mesmo Raspberry Pi como dispositivo IoT. Esse segundo experimento tem como objetivo avaliar o MLOOF em um ambiente real que pode sofrer interferências em relação a qualidade da rede, latências entre dispositivos e servidores e utilização dos servidores por outros usuários.

3. **Experimento Simulado. Objetivo: verificação de diversidade e escalabilidade:** O último experimento utiliza um simulador em software. A simulação nos permite realizar experimentos que seriam impossíveis de serem feitos em ambiente real com tempo limitado. Podemos avaliar o sistema em situações com distintas características de rede, dispositivos diversos, carga de trabalho variados e outros aspectos dinâmicos que podemos variar de forma controlada.

4.1 Cenários de teste comuns

Diferentes tipos de aplicações com comportamento de execução distintos sofrem impactos diferentes quando executados em uma plataforma remota. Durante o processo de *offloading*, uma aplicação que executa muito processamento em uma pequena quantidade de dados não é tão afetada por uma conexão ruim entre dispositivo e servidor, mas é afetada caso o servidor tenha baixo poder de processamento. De forma semelhante, uma aplicação que trabalha com uma quantidade grande de dados e pouco processamento precisa de uma boa qualidade de rede mas não necessita de um servidor muito poderoso. Por esses motivos, os aplicativos desenvolvidos para os experimentos foram projetados para terem diferentes combinações de carga de processamento e volume da dados a transmitir.

Tabela 4.1. Cenários de experimentação, para execução local, na *cloudlet* e na nuvem

Cenários	Carga de processamento	Volume de dados (parâmetros e resultados)	Transações de Rede
A	Variável	Baixo	Poucas
B	Baixa	Variável	Poucas
C	Baixa	Baixo	Variável

Para possibilitar a avaliação do MLOOF, foram definidos cenários de teste que variam a carga de processamento, o volume de dados transmitidos e quantidade de transações de rede. Os cenários de teste permitem avaliar os impactos do *offloading* e o benefício causado pela *cloudlet* no sistema. Foram comparados tempo de execução e consumo energético de duas aplicações com características distintas (uma consome muito processamento, e outra trabalha com muitos dados). As aplicações são executadas localmente nos dispositivos, realizando o *offloading* para uma *cloudlet* e realizando o *offloading* para um servidor na nuvem. A Tabela 4.1 apresenta os cenários escolhidos.

Esses cenários foram escolhidos de forma a possibilitar a avaliação isolada do impacto causado por duas características principais das aplicações que utilizam técnicas de *offloading*, carga de processamento e volume de dados transmitidos pela rede. Analisando essas características de forma separada, conseguimos entender melhor como o desempenho do MLOOF por ser afetado por aspectos específicas das aplicações.

Para a execução do cenário de testes A (cenário que requer alta carga de processamento mas baixo volume de dados transmitidos), foi desenvolvido um aplicativo que executa o algoritmo de Fibonacci recursivo. O algoritmo recebe como entrada um número inteiro positivo e calcula o número de Fibonacci de acordo com o seguinte algoritmo:

Algorithm 3: Algoritmo Fibonacci Recursivo

```
1 Function Fibonacci(N):
2     if N <= 1 then
3         Retorna N;
4     else
5         Retorna Fibonacci(N - 1) + Fibonacci(N - 2);
6     end
7 return
```

Para a execução do *offloading* desse aplicativo, é necessário enviar poucos bytes de informação pela rede, que identifica o número de Fibonacci a ser calculado, porém a carga de processamento necessária pode ser grande, por se tratar de um algoritmo com

custo exponencial. A complexidade assintótica desse algoritmo é, aproximadamente, $\theta(1.6^n)$.

Programas que precisam realizar cálculos matemáticos e científicos (e utilizem poucos dados para tais cálculos) são exemplos de aplicações reais com características similares ao cenário A.

O cenário de testes B apresenta baixa carga de processamento em uma grande quantidade de dados que devem ser transmitidos pela rede durante o processo de *offloading*. Para realizar os experimentos, foi desenvolvida uma aplicação que transforma uma imagem colorida em outra imagem em escala de cinza. A imagem original é enviada ao servidor (*cloudlet* ou nuvem), que aplica o filtro de escala de cinza e devolve uma imagem filtrada ao dispositivo. O algoritmo de filtragem passa uma vez por cada pixel da imagem substituindo-o pela média de suas componentes R, G e B, que indicam a quantidade de vermelho, verde e azul, respectivamente, de cada pixel. Este algoritmo tem complexidade linear no tamanho da imagem e a imagem deve ser transmitida pela rede duas vezes durante o processo de *offloading*, uma vez para a imagem colorida ser enviada ao servidor e uma segunda vez para a imagem em escala de cinza ser enviada para o cliente. A aplicação funciona de acordo com o seguinte algoritmo:

Algorithm 4: Algoritmo Escala de Cinza

```
1 Function EscalaDeCinza(I):
2     foreach Pixel p{r,g,b} ∈ I do
3         c = (r + g + b)/3;
4         p = {c, c, c};
5     end
6 return
```

Alguns algoritmos de processamento digital de imagem e vídeo são exemplos de aplicações com características similares ao cenários de testes B.

O cenário de testes C apresenta uma aplicação que requer baixa carga de processamento, envia e recebe baixo volume de dados pela rede mas faz vários pedidos de *offloading* em sequência, indicando indicando alta quantidade de transações de rede. São exemplos de aplicações que possuem essas características aquelas que possuem restrições de tempo real, nas quais o resultado de uma operação é útil para a aplicação apenas durante um curto período de tempo.

Para realizar os experimentos foi utilizado o mesmo algoritmo de Fibonacci usado no cenário A. O número de Fibonacci a ser calculado foi fixado em um valor específico e a aplicação realiza diversos pedidos de *offloading* em sequência utilizando esse mesmo valor.

Aplicações com restrições de execução em tempo real possuem características similares ao cenário de testes C. Por exemplo, algoritmos para processamento de imagem em tempo real e vídeo games fazem parte deste grupo de aplicações.

Esses três cenários descritos acima foram utilizados em todos os experimentos que estão descritos nas próximas seções.

4.2 Experimento em Ambiente Controlado

Este primeiro experimento tem como objetivo verificar a funcionalidade do sistema e realizar uma primeira análise de desempenho do MLOOF. Os testes foram feitos utilizando um Raspberry PI como dispositivo cliente e computadores pessoais como servidores *cloudlet* e da nuvem. As condições para o experimento foram bem controladas, sendo que os servidores estavam sendo utilizados exclusivamente para os testes e a rede local não estava sendo compartilhada com outros dispositivos. Com os resultados obtidos podemos preparar experimentos em ambientes mais realistas que possam apresentar resultados mais condizentes com os que observaríamos em uma instalação em ambiente de produção.

4.2.1 Metodologia

Para a execução dos experimentos foram utilizados um dispositivo IoT, um servidor *cloudlet* e um servidor na nuvem. O dispositivo cliente pode se conectar diretamente ao servidor da *cloudlet* via Wi-Fi porque estão localizados na mesma rede local. Outra opção é se conectar diretamente ao servidor da nuvem via internet. A Figura 4.1 apresenta um diagrama da topologia utilizada durante o experimento.

Após a montagem do dispositivo e dos servidores, foram iniciados os testes. Os testes consistem na execução de aplicações pelo dispositivo cliente e no processo de *offloading* para os servidores. Foram escolhidas aplicações com características distintas para permitir a avaliação do impacto dessas características no desempenho do sistema. Estas características são: (1) Carga de processamento, para o qual foi implementada uma aplicação que computa um número de Fibonacci utilizando o algoritmo recursivo, o que nos permite variar significativamente a utilização de CPU necessária para a aplicação; (2) Volume de dados transmitidos pela rede, para o qual foi utilizada uma aplicação que realiza a transformação de uma imagem colorida em sua respectiva imagem em escala de cinza, e essas imagens devem ser transmitidas pela rede durante o processo de *offloading*; (3) Quantidade de transações de rede, onde a mesma aplicação que calcula o número de Fibonacci foi configurada para realizar diversos pedidos

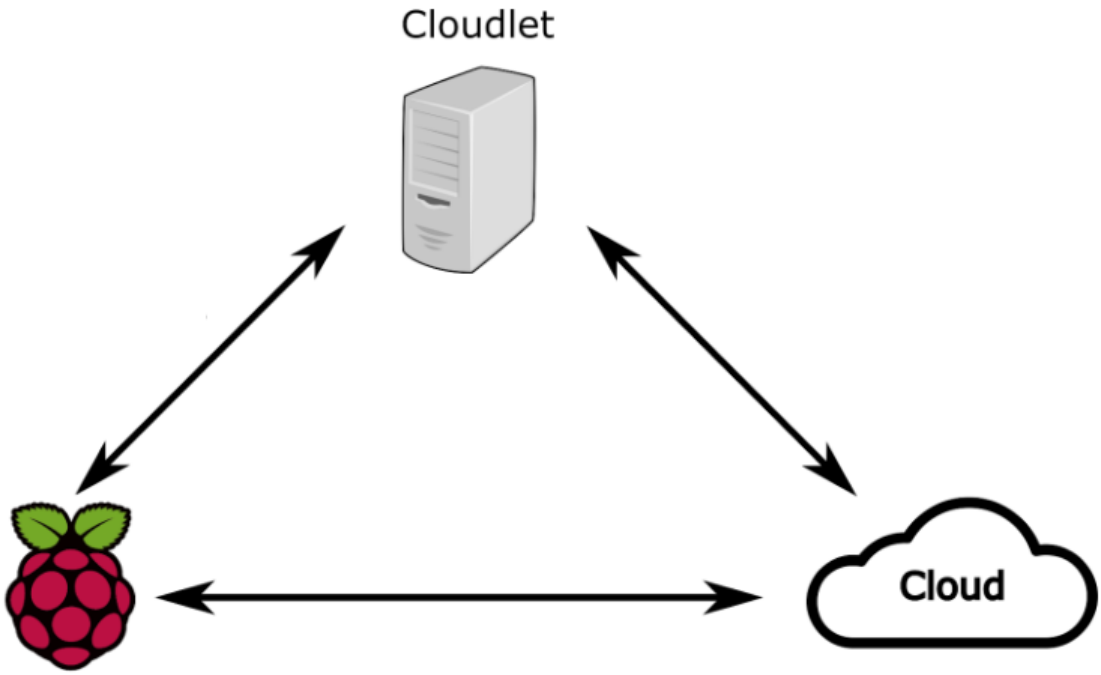

Figura 4.1. Diagrama de topologia com um dispositivo, um servidor *cloudlet* e um servidor na nuvem.

de *offloading* em sequência. Os cenários de teste utilizados estão descritos em maior detalhe na Seção 4.1.

O objetivo dos testes é avaliar o desempenho do sistema durante a execução da aplicação nos três ambientes de execução: o dispositivo IoT, o servidor da *cloudlet* e o servidor da nuvem. As aplicações foram executadas variando seus parâmetros progressivamente e as métricas avaliadas foram tempo de processamento e consumo energético. O tempo de processamento é o tempo total de processamento e inclui o tempo de transmissão e recebimento dos dados no caso da realização do *offloading*. O consumo energético é medido no dispositivo cliente e depende da carga de processamento durante a execução local da aplicação e do consumo de energia proveniente da transmissão dos dados pela rede durante a execução remota.

4.2.2 Montagem e Execução

Para execução dos experimentos de avaliação da proposta, foi utilizado como dispositivo móvel o microcontrolador Raspberry Pi 3 Model B v1.2. Ele possui um processador Quad Core 1.2GHz Broadcom BCM2837 de 64 bits, 1GB de memória RAM, conexão de internet *Wi-Fi* e *Ethernet*, e roda *Raspbian*, um sistema operacional baseado na distribuição *Debian* do Linux. Todos os experimentos foram repetidos trinta vezes e

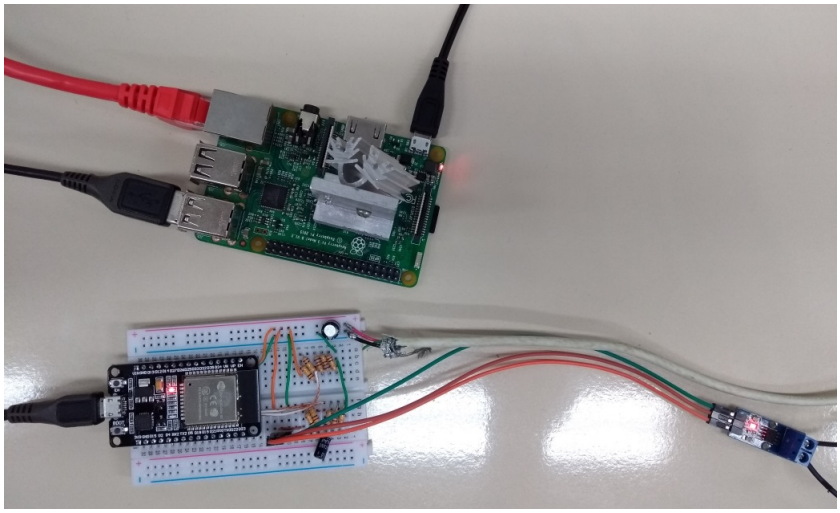

Figura 4.2. Arranjo para medição de energia consumida (parte inferior da figura). Dispositivo Raspberry Pi (parte superior da figura).

os resultados apresentados são a média das repetições.

Para realizar medições de consumo energético do Raspberry Pi foi criado um hardware separado que, quando colocado entre a fonte de energia e o Raspberry, realiza medições instantâneas de tensão e corrente. Os dados coletados são processados por um microcontrolador e transferidos para outra máquina via interface USB para análise. Esse hardware foi criado utilizando-se um kit de desenvolvimento NodeMCU equipado com o microcontrolador ESP-WROOM-32 e um sensor de corrente que emprega o circuito integrado ACS712ELCTR-05B-T. A Figura 4.2 mostra, na parte inferior, o arranjo montado para medição de energia consumida no dispositivo Raspberry Pi, representado na parte superior da imagem.

No papel de *cloudlet* foi utilizado um laptop que possui um processador Intel Core i3-4005U 1.7GHz, 4GB de memória RAM, rodando o sistema operacional Ubuntu versão 16.04. A *cloudlet* está conectada via *Wi-Fi* à mesma rede local do Raspberry, a um *hop* de distância. No papel de nuvem foi utilizado um PC com processador AMD FX-4300 3.80GHz, 8GB de memória RAM, rodando o sistema operacional Ubuntu versão 16.04. A nuvem está conectada à internet via interface *Ethernet*, e se encontra a quinze *hops* de distância do dispositivo IoT. Durante os experimentos, a latência entre dispositivo e nuvem foi de 170ms em média, e a latência entre dispositivo e *cloudlet* foi de 9ms em média.

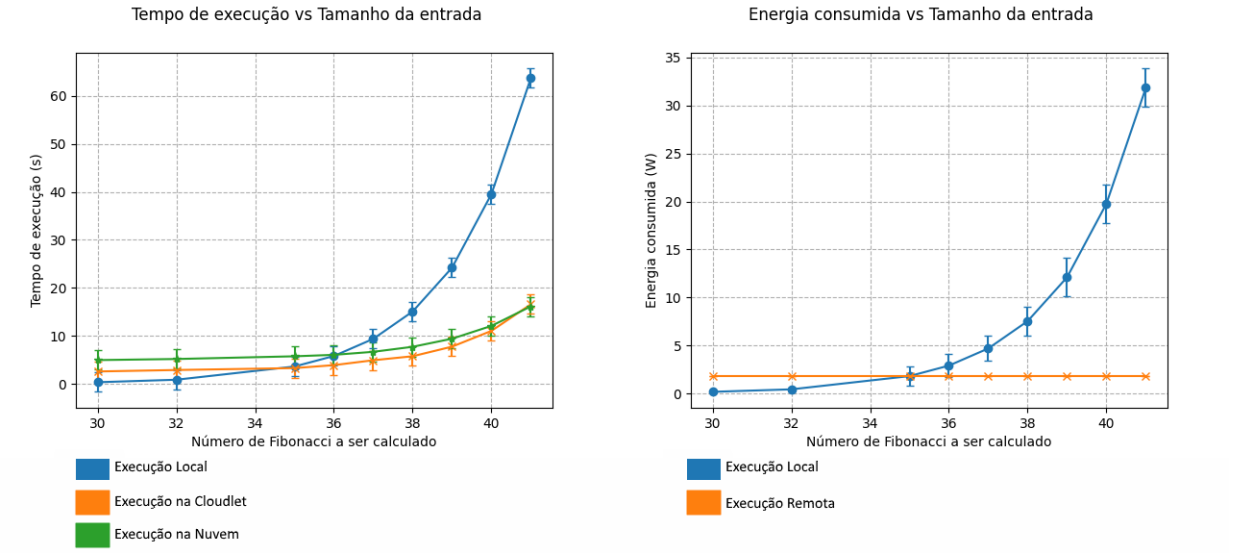

Figura 4.3. Gráficos de resultados dos experimentos no cenário de testes A.

4.2.3 Resultados e Avaliação

A Figura 4.3 apresenta os resultados dos testes feitos para o cenário de testes A. O gráfico à esquerda apresenta as médias de tempo de execução do método quando executado localmente e quando há o *offloading* para o servidor da *cloudlet* e para o servidor da nuvem. Como o tempo de processamento do algoritmo aumenta com o aumento do valor da entrada, a energia consumida pelo dispositivo durante a execução local também aumenta. Com a execução remota, porém, a energia consumida se mantém constante, porque não há variação significativa da quantidade de dados transmitidos pela rede quando a entrada do algoritmo varia. O gráfico à direita da Figura 4.3 apresenta as médias de consumo energético do dispositivo durante a execução remota e local do método.

Para calcular números de Fibonacci pequenos (menores do que 35, nos testes realizados) é mais interessante executar a computação no próprio dispositivo, porque tanto o tempo de execução quanto o consumo energético são menores quando comparados ao tempo e consumo energético de enviar os dados pela rede e aguardar o resultado. Para valores maiores, o consumo energético da computação local supera o consumo de enviar os dados pela rede, e o tempo de execução no hardware do Raspberry supera o tempo de execução no hardware dos servidores mais o tempo de comunicação entre eles. Nos testes realizados, o tempo de execução no dispositivo chega a ser quatro vezes maior do que o tempo de execução remoto, e a energia consumida chega a ser dezessete vezes maior. Para calcular números de Fibonacci maiores, essa diferença seria ainda maior. Ao optar por realizar o *offloading* para *cloudlets*, o tempo de execução chega

a ser 49,4% menor do que ao realizar o *offloading* para o servidor da nuvem. Para números de Fibonacci muito grandes, o servidor da nuvem supera o tempo de execução da *cloudlet*, por se tratar de uma máquina com maior poder de processamento. Naturalmente, esses valores dependem das capacidades e disponibilidades dos servidores remotos: quanto maior a capacidade, melhor o resultado.

O gráfico à esquerda da Figura 4.4 apresenta os resultados dos testes feitos para o cenário de testes B. O gráfico apresenta as médias de tempo de execução ao realizar o *offloading* para a *cloudlet* e para a nuvem, ao variar o tamanho dos dados transmitidos representando imagens. O ponto fundamental desse experimento é o tempo de transmissão das imagens, que é menor para o servidor da *cloudlet* por estar na mesma rede local do Raspberry. O tempo de execução local, bem como o o tempo de processamento dos dados nos servidores, é irrelevante se comparado ao tempo de execução remoto, considerando transmissão e processamento, e por isso não foi apresentado no gráfico. O tempo de execução remoto é, em grande parte, devido à utilização da rede. A quantidade de dados transmitidos mostrada no gráfico representa o tamanho da imagem transmitida, isso significa que, contando com a ida e a volta dos dados, o dispositivo transmite e recebe o dobro do valor mostrado no gráfico.

O gráfico à direita da Figura 4.4 apresenta o tempo de execução de diversas chamadas consecutivas do algoritmo de Fibonacci (cenário C). Vários aplicativos com restrições de tempo real necessitam executar métodos em intervalos curtos de tempo, dependendo de suas restrições. Utilizando *cloudlets* no lugar de servidores na nuvem temos ganhos de tempo significativos (até 74% de ganho nos testes realizados), o que pode ser a diferença entre possibilitar ou não a utilização de técnicas de *offloading* nessas aplicações.

4.3 Experimento com Testbed

Este segundo experimento tem como objetivo avaliar o desempenho do MLOOF em um ambiente real. Para isso foi utilizada a infraestrutura do *testbed* FUTEBOL [UFMG, 2020a] como servidor *cloudlet* e um servidor Amazon AWS como servidor da nuvem. Algumas características do primeiro experimento foram mantidas, como o Raspberry Pi que foi utilizado como dispositivo IoT e a topologia de rede, apresentada pela figura 4.1.

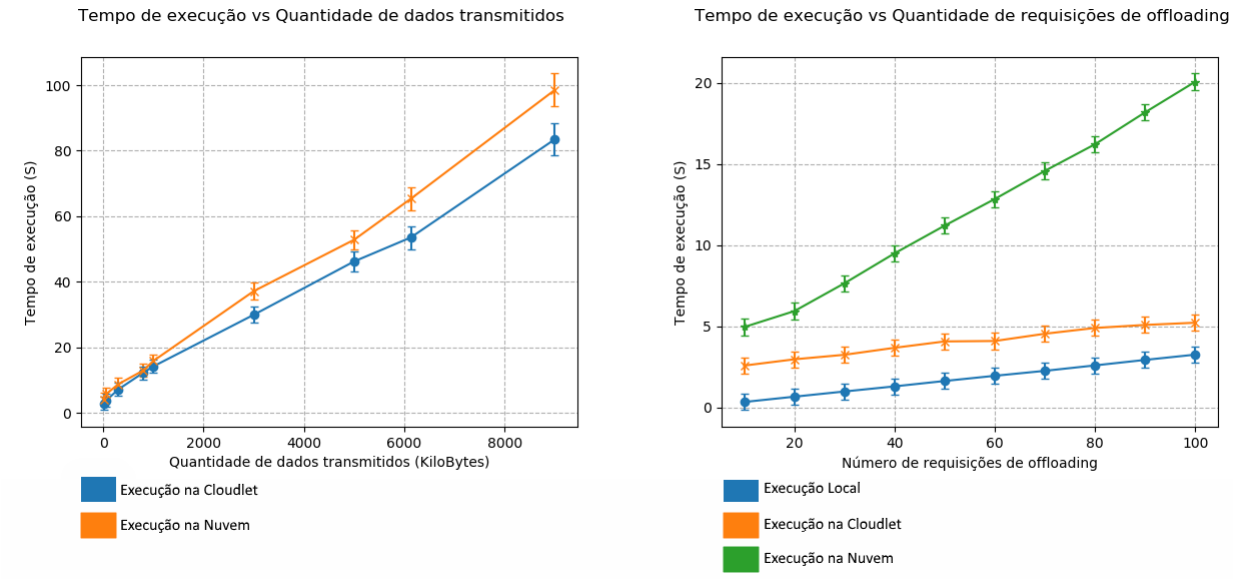

Figura 4.4. Gráficos de resultados dos experimentos nos cenários de testes B e C.

4.3.1 Metodologia

Para a execução dos experimentos foram utilizados um dispositivo IoT, um servidor *cloudlet* e um servidor na nuvem. O dispositivo cliente pode se conectar diretamente ao servidor da *cloudlet* via Wi-Fi porque estão localizados na mesma rede local. Outra opção é se conectar diretamente ao servidor da nuvem via internet.

As principais diferenças entre esse experimento com o experimento apresentado na seção 4.2 são as características de rede e dos servidores utilizados. Como servidor *cloudlet* foi utilizado um servidor do *testbed* FUTEBOL [UFMG, 2020a] e como servidor da nuvem foi utilizado um servidor Amazon AWS.

Após a montagem dos dispositivos e inicialização virtual dos servidores, foram iniciados os testes. A inicialização em ambos os servidores é feita virtualmente através de suas respectivas interfaces de comunicação. O servidor da *cloudlet* pode ser acessado através da aplicação *jFed* [UFMG, 2020b]. Essa aplicação faz parte da plataforma *Fed4Fire*, uma plataforma aberta que permite a utilização de uma variedade de *testbeds* nas áreas de pesquisas científicas.

A metodologia adotada, incluindo os cenários de testes escolhidos, foi semelhante à utilizada no primeiro experimento e pode ser lida com mais detalhes na seção 4.2.

4.3.2 Montagem e Execução

Para a execução dos experimentos foi utilizado como dispositivo móvel o mesmo Raspberry Pi utilizado no experimento em ambiente real. Todos os experimentos foram repetidos trinta vezes e os resultados apresentados são a média das repetições. As medições de energia foram realizadas utilizando os dados obtidos com o mesmo hardware criado no primeiro experimento.

No papel de *cloudlet* foi utilizada uma máquina virtual padrão do *testbed* FUTEBOL. Após a criação da máquina virtual, o código binário do servidor foi transferido e o servidor foi inicializado, estando pronto para receber requisições de clientes. No papel de servidor na nuvem foi utilizado um servidor virtual da Amazon AWS de propósito geral. O servidor utilizado foi um *t2.micro*, composto por um CPU virtual e um GiB de memória RAM, rodando um sistema operacional Linux de 64 bits. Durante os experimentos, a latência entre dispositivo e nuvem foi de 130ms em média, e a latência entre dispositivo e *cloudlet* foi de 9ms em média. A topologia final para esse experimento é apresentada na Figura 4.5. A figura representa a distância utilizando a métrica *hops*, que indica por quantos elementos de rede os pacotes de dados precisaram passar para chegar ao destino. Os valores foram obtidos utilizando a ferramenta *traceroute* do Linux.

4.3.3 Resultados e Avaliação

A Figura 4.6 apresenta os resultados dos testes feitos para os cenários de testes A e B. O primeiro gráfico apresenta os resultados do cenário de testes A. O resultado indica as médias de tempo de execução do método quando executado localmente e quando há o *offloading* para o servidor da *cloudlet* e para o servidor da nuvem. Constatamos que os resultados se comportam de forma semelhante aos resultados obtidos no primeiro experimento. O tempo de execução local é o menor para números de Fibonacci muito pequenos mas se torna muito maior do que o tempo de execução remota em ambos os servidores a partir de certo ponto (no caso dos testes, para números de Fibonacci maiores que 36). A comparação entre os servidores também se mostra semelhante, o tempo de execução no servidor da nuvem é maior para números de Fibonacci menores devido à maior latência, mas se torna menor para números maiores, onde o tempo de processamento é muito maior do que a latência, diminuindo seu impacto.

O segundo gráfico apresenta os resultados dos testes feitos para o cenário de testes B. O gráfico apresenta as médias de tempo de execução ao realizar o *offloading* para a *cloudlet* e para a nuvem, ao variar o tamanho dos dados transmitidos representando imagens. O ponto fundamental desse experimento é o tempo de transmissão

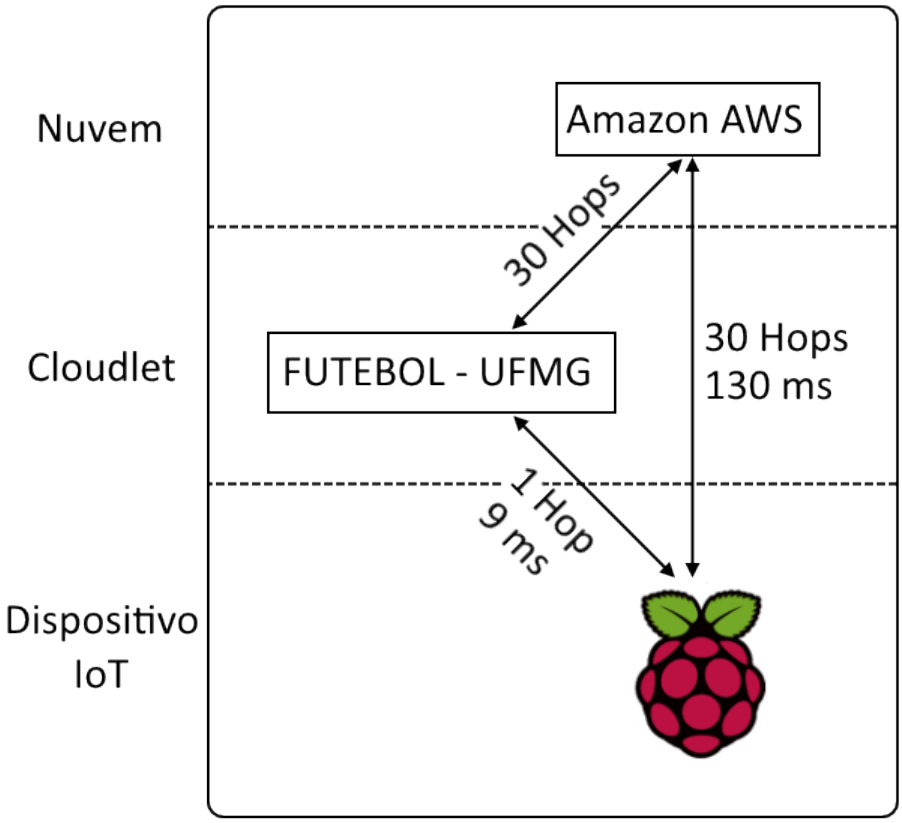

Figura 4.5. Diagrama de topologia utilizada no experimento com testbed.

das imagens, que é menor para o servidor da *cloudlet* por estar na mesma rede local do Raspberry. O tempo de execução local, bem como o o tempo de processamento dos dados nos servidores, é irrelevante se comparado ao tempo de execução remoto, considerando transmissão e processamento, e por isso não foi apresentado no gráfico. O tempo de execução remoto é, em grande parte, devido à utilização da rede. Analogamente ao experimento em ambiente controlado, a quantidade de dados transmitidos mostrada no gráfico representa o tamanho da imagem transmitida. A quantidade de dados transmitida somada com a quantidade de dados recebida é o dobro do valor mostrado.

A Figura 4.7 apresenta os resultados dos testes feitos para os cenários de testes C. A aplicação nesses experimentos realiza diversas chamadas consecutivas do algoritmo de Fibonacci (mesmo algoritmo utilizado no cenário A). Vários aplicativos com restrições de tempo real necessitam executar métodos em intervalos curtos de tempo, dependendo de suas restrições. Utilizando *cloudlets* temos ganhos de tempo significativos justamente por possuir latência bem menor quando comparada à latência de comunicação do servidor da nuvem (9ms versus 130ms).

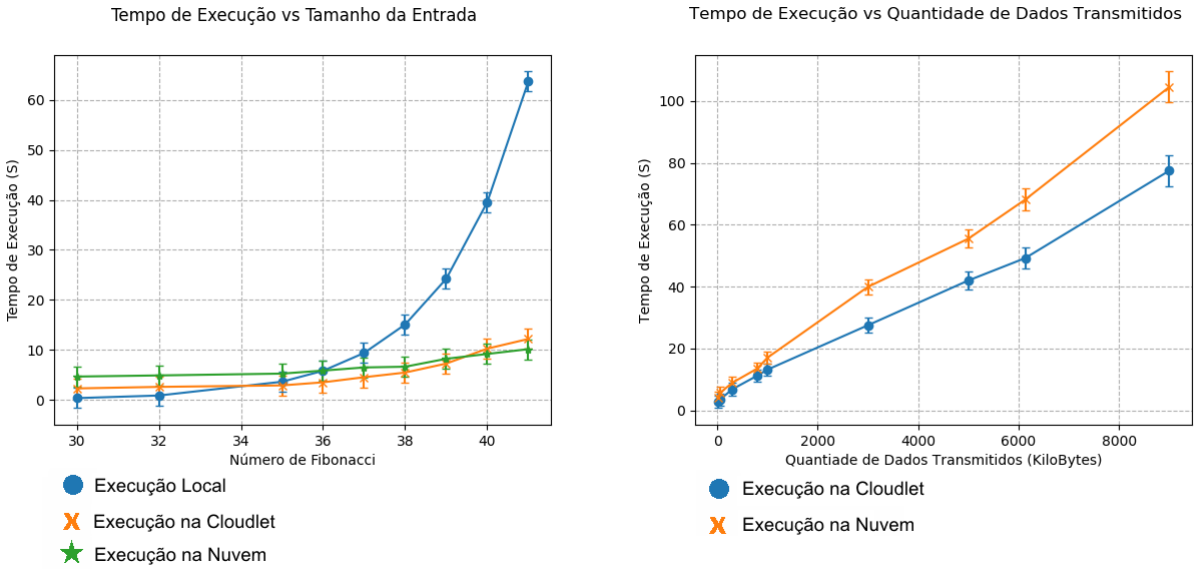

Figura 4.6. Casos de uso A e B do experimento com testbed. Gráfico à esquerda apresenta os resultados para o caso de uso A e o gráfico à direita apresenta os resultados para o caso de uso B.

4.4 Experimento Simulado

O terceiro experimento foi executado em um simulador e teve como objetivo avaliar a escalabilidade e o desempenho do MLOOF em dispositivos, servidores e redes com características distintas dos que foram avaliados nos testes anteriores. A simulação nos permite realizar experimentos que seriam muito difíceis ou mesmo impossíveis de serem feitos em ambiente real, levando em conta todas as limitações que temos. Podemos avaliar o sistema em situações com distintas características de rede, diversidade de dispositivos, carga de trabalho variados e outros aspectos dinâmicos que podemos variar de forma controlada.

4.4.1 Simulador

Para executar os estudos de simulação, utilizamos um simulador da literatura. O IFogSim [Mahmud & Buyya, 2019] é um kit de desenvolvimento de simulações desenvolvido em Java de alto desempenho para simulações de *fog computing*, *edge computing* e IoT. O simulador modela ambientes IoT e simula a administração de recursos em termos de latência, congestionamento de rede, tempo de execução e consumo de energia em diferentes cenários. As classes do simulador permitem a definição de dispositivos, sensores, atuadores, aplicações e serviços de gerenciamento de recursos.

Outros simuladores foram analisados, mas o IFogSim foi escolhido por se tra-

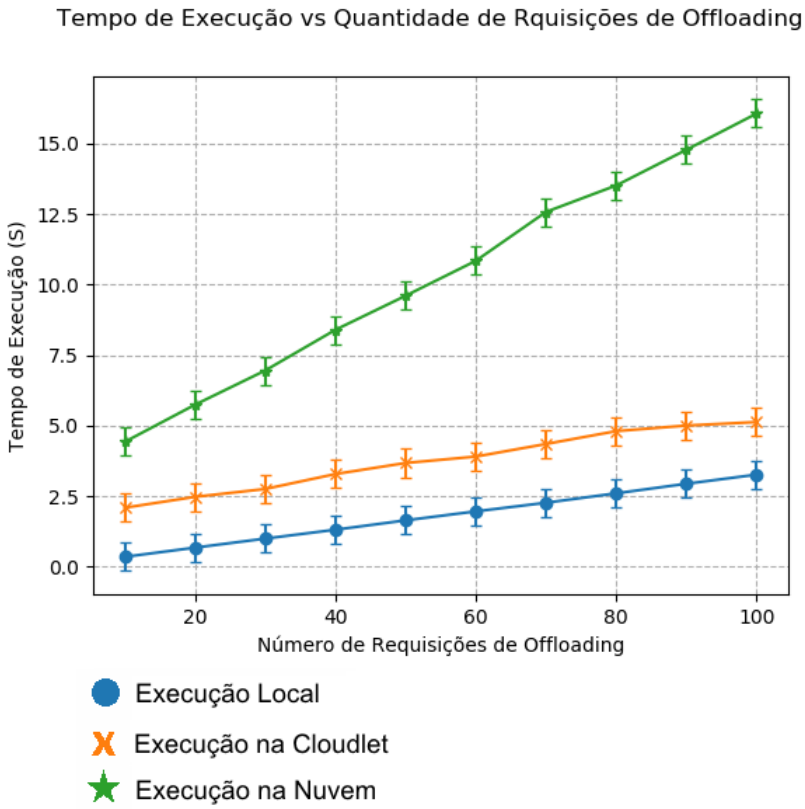

Figura 4.7. Gráfico apresentando os resultados obtidos no experimento com testbed no caso de uso C.

tar de um simulador bem consolidado, escrito na mesma linguagem de programação do MLOOF, o que facilita a integração dos sistemas, atualmente em uso em diversos trabalhos. Ele possui código aberto, o que facilita a realização de modificações necessárias. Apesar de estar disponível há algum tempo, o simulador continua sendo utilizado em trabalhos envolvendo arquiteturas de *fog/edge computing* [Al-Khafaji et al., 2019], [Toor et al., 2019], [Qasem et al., 2020].

4.4.2 Metodologia

A metodologia consiste das seguintes atividades:

1. **Cenários de teste:** Para executar as simulações, inicialmente são feitas as descrições de todos os elementos do sistema: dispositivos, servidores, topologia e características de rede, aplicação e serviços de gerenciamento de recursos. Com todas as características definidas, ao executar o simulador obtemos informações de tempo de execução das aplicações, consumo energético dos dispositivos e quaisquer outras características medidas, como custo de servidores ou utilização de

rede. As características das aplicações utilizadas na simulação são definidas de acordo com o cenário de teste desejado.

2. **Configuração:** As descrições dos elementos do sistema baseadas no cenário de teste a ser executado devem ser configuradas no simulador.

3. **Validação:** As primeiras simulações executadas fazem parte do processo de validação. Quando comparamos os resultados gerados pelo simulador com resultados previamente obtidos em experimentos reais, verificamos que as características das aplicações no simulador foram as mesmas características dos cenários definidos na seção 4.1. Foram criados três cenários com objetivo de avaliar o desempenho do sistema, para isso variando a carga de processamento requerida pela aplicação, o volume de dados transmitidos pela rede no caso de execução remota e o número de transações de rede (cenários A, B e C respectivamente).

4. **Execução das simulações:** Após o processo de validação ser concluído com sucesso, o que indica que o simulador foi configurado de forma a representar corretamente o comportamento esperado de um sistema em ambiente real, foram feitas simulações com casos ainda não testados com dispositivos físicos. Foram criados casos de uso que fariam sentido existir em um sistema no mundo real mas que não foram realmente empregados por limitações de tempo e custo. Com esses casos de uso podemos definir as características dos dispositivos e servidores, além da topologia de rede e características das aplicações que possam ser implementadas no simulador.

5. **Análise e avaliação dos resultados:** Após a execução de cada simulação, realizamos a análise e avaliação dos resultados, entendendo como o sistema se comportou e qual foi o impacto da utilização do MLOOF em cada cenário.

4.4.3 Validação

O primeiro passo da simulação consiste em realizar a validação dos resultados obtidos pelo simulador, utilizando os dados previamente obtidos nos experimentos em ambiente real. Para isso, as características do dispositivo IoT e dos servidores foram copiadas para o simulador e as aplicações também foram importadas. Como todo o sistema está escrito em Java e o simulador também é feito na mesma linguagem, as importações são realizadas sem necessidade de qualquer modificação. Os casos de teste utilizados para a verificação são os mesmos utilizados durante os experimentos em ambiente real (descritos com mais detalhe na seção 4.1).

Primeiramente é feita a configuração do simulador com informações dos mesmos dispositivos, servidores, aplicações e topologia utilizados nos experimentos reais. Os dados do dispositivo cliente foram retirados das especificações técnicas do microcontrolador Raspberry Pi 3 Model B v1.2, que possui um processador Quad Core 1.2GHz Broadcom BCM2837 de 64 bits e 1GB de memória RAM. Como servidor *cloudlet* os dados vieram de um processador Intel Core i3-4005U 1.7GHz com 4GB de memória RAM, e como servidor da nuvem um processador AMD FX-4300 3.80GHz com 8GB de memória RAM. Esses dados foram escolhidos dessa forma para que as configurações sejam idênticas às utilizadas no primeiro experimento em ambiente real. A organização e conexão entre dispositivo e servidores pode ser observada na Figura 4.1.

Após a configuração do simulador, os experimentos são executados e, como resultado, obtemos dados de consumo energético, tempo de processamento, quantidade de dados transmitidos, dentre outras informações sobre os recursos utilizados no processo.

O gráfico da Figura 4.8 apresenta a comparação dos resultados obtidos para o primeiro caso de teste (tempo de execução do experimento em ambiente real em azul e tempo de execução obtido no simulador em alaranjado). Os resultados são compatíveis e variam no máximo, aproximadamente, 10%. Isso se deve a variações obtidas durante o experimento em ambiente real que são difíceis de controlar, e possíveis configurações do simulador que não se adéquam perfeitamente às condições existentes em um experimento real.

Os dados obtidos na simulação de avaliação indicam que o simulador está bem configurado para podermos explorar novas possibilidades que ainda não foram avaliadas nos experimentos reais. As próximas seções contém os detalhes de planejamento e resultados obtidos com as novas simulações.

4.4.4 Casos de Uso

O objetivo de fazer os experimentos simulados é extrapolar o que nós já conhecemos da realidade e aplicar esse conhecimento em situações de difícil replicação no mundo real. Essas situações podem apresentar difícil replicação devido a altos custos, arquitetura complexa ou limitações de tempo. Até agora realizamos experimentos em ambiente real e validação do ambiente simulado. Os casos de uso apresentados a seguir representam situações onde o MLOOF pode ser útil e podem ser implementados em nosso simulador.

As seguintes características do sistema, rede ou dispositivos sofreram variações durante as simulações:

1. **Capacidade e variedade dos dispositivos e servidores:** O objetivo principal do MLOOF é diminuir o tempo de execução de aplicações e/ou o consumo

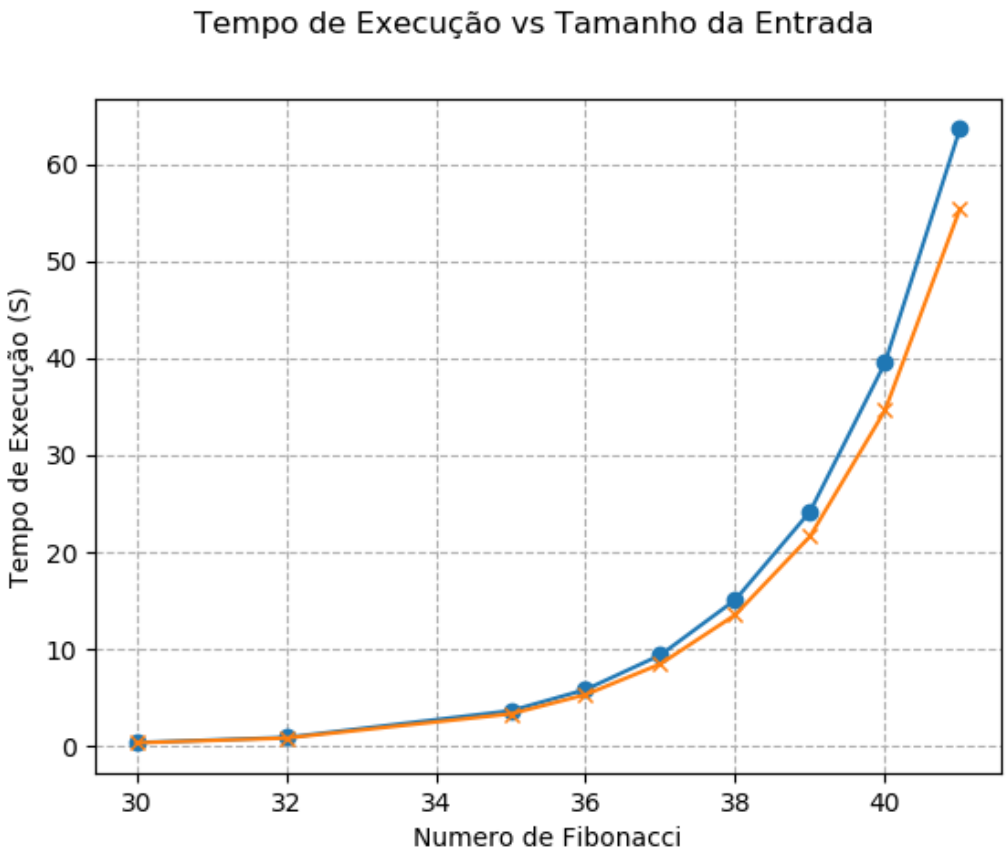

Figura 4.8. Gráfico apresentando a comparação de resultados obtidos em experimento real (linha azul) e experimento simulado (linha alaranjada).

energético dos dispositivos, o que torna sua utilização bastante atraente para dispositivos móveis com baixa capacidade computacional. Os experimentos realizados nas seções anteriores utilizaram um dispositivo com capacidade computacional maior do que grande parte dos dispositivos IoT. A simulação nos permite avaliar os ganhos do nosso arcabouço em dispositivos mais fracos e em servidores mais carregados.

2. **Quantidade de dispositivos e servidores:** Utilizando o simulador podemos ampliar o número de dispositivos e avaliar a escalabilidade do sistema.

3. **Características de rede:** Em ambientes reais ficamos limitados a apenas uma ou poucas topologias de rede. A topologia da rede e a distribuição geográfica dos dispositivos e servidores influencia as características da rede e, consequentemente, o desempenho do MLOOF.

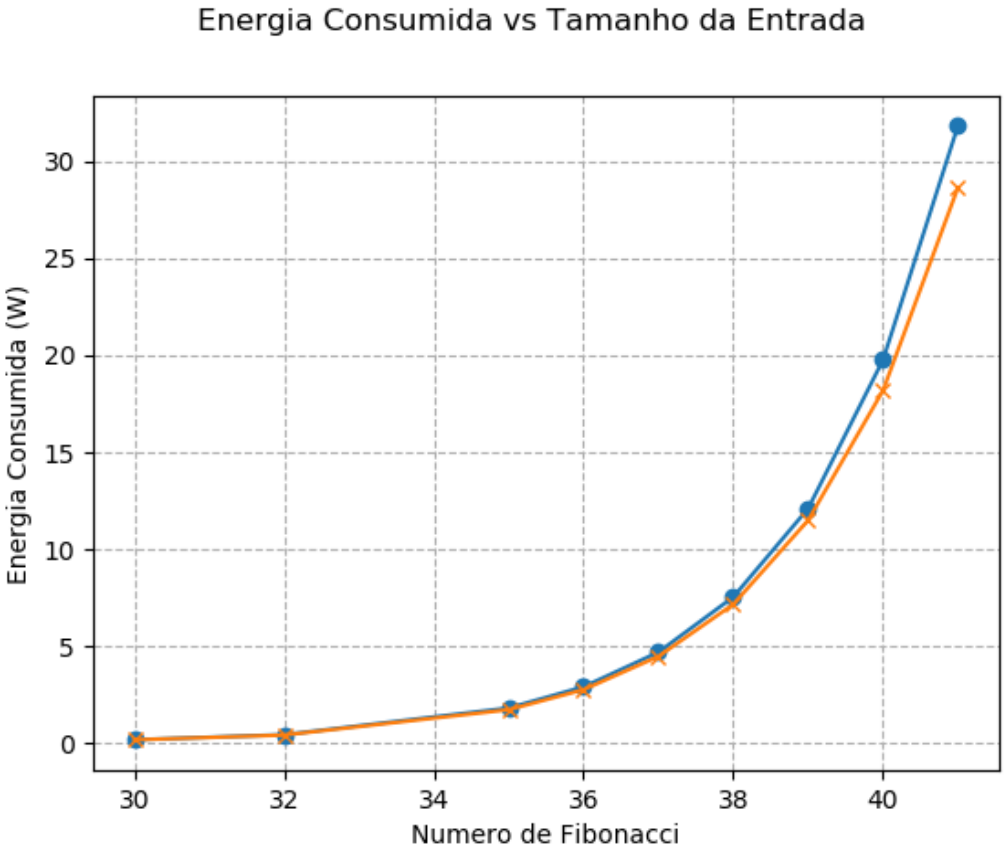

Figura 4.9. Gráfico apresentando a comparação de resultados obtidos em experimento real (linha azul) e experimento simulado (linha alaranjada).

4.4.5 Cenários de Teste de Simulação

1. **Primeiro teste: Capacidade dos dispositivos e servidores:** Teste mudando apenas o dispositivo utilizado nos testes reais para um dispositivo mais simples. Mudança de Raspberry Pi 3 para Arduino Uno, que é um micro controlador com muito menos poder de processamento quando comparado ao Raspberry.

2. **Segundo teste: Escalabilidade:** Aumentar o número de dispositivos mantendo a quantidade de servidores. Este teste tem como objetivo verificar como os servidores podem influenciar no desempenho do sistema quando estão sobrecarregados com alta demanda dos clientes.

3. **Terceiro teste: Variações de topologias:** Modificar as topologias, fazendo com que tenhamos diferentes valores de vazão e latência na rede para diferentes pares dispositivo-servidor.

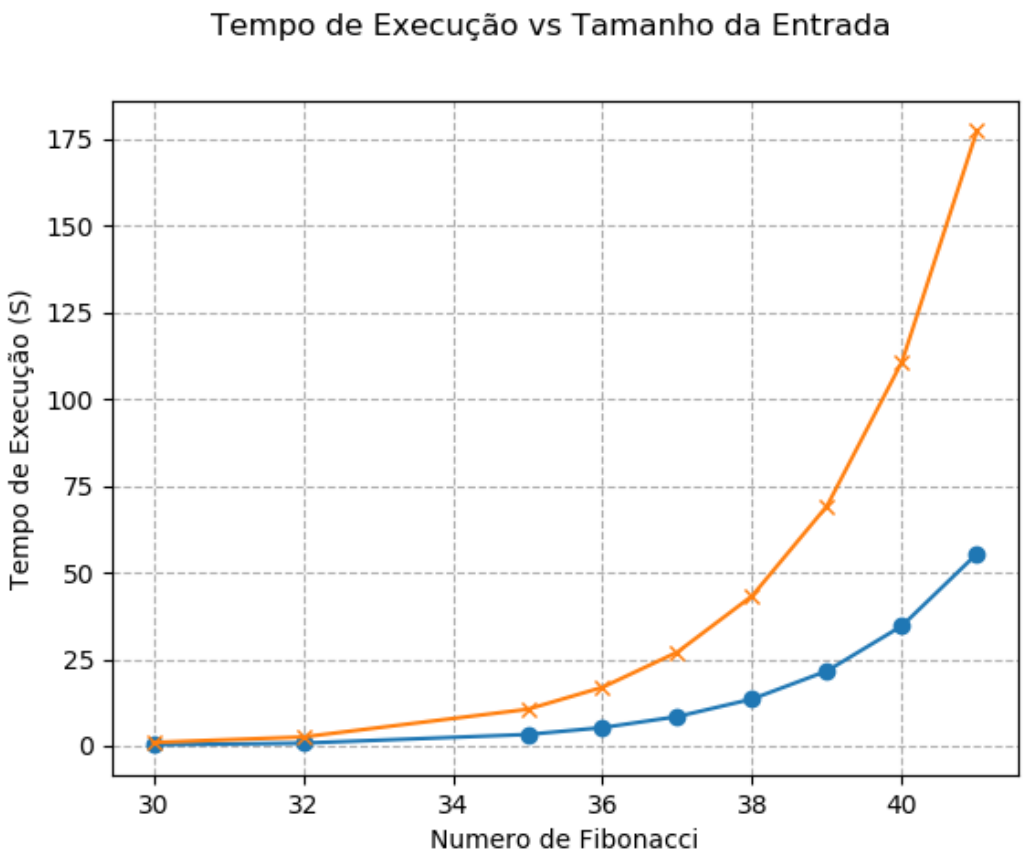

Figura 4.10. Comparação de resultados obtidos em simulação com Raspberry Pi (linha azul) e Arduino Uno (linha alaranjada) com aplicação rodando localmente.

4.4.6 Primeiro teste: Capacidade dos dispositivos e servidores

Como primeiro caso de teste, avaliamos o desempenho do nosso arcabouço com dispositivos e servidores de diferentes capacidades de processamento. Nos testes em ambiente real pudemos verificar a eficiência do MLOOF rodando em apenas um dispositivo móvel. Aqui estamos avaliando o MLOOF em um dispositivo IoT mais fraco quando comparado ao Raspberry Pi utilizado anteriormente, o que é mais condizente com a realidade do poder de processamento dos dispositivos IoT.

O dispositivo utilizado na simulação foi um Arduino Uno. Utilizamos as mesmas aplicações que foram utilizadas nos primeiros experimentos em ambiente real e servidores com mesmo poder computacional. Os resultados podem ser observados no gráfico da Figura 4.10.

Por se tratar de um dispositivo com menos poder de processamento, era esperado que o Arduino Uno gastaria mais tempo para executar os algoritmos quando comparado ao Raspberry. A Figura 4.11 apresenta a comparação dos tempos de execução ao rodar

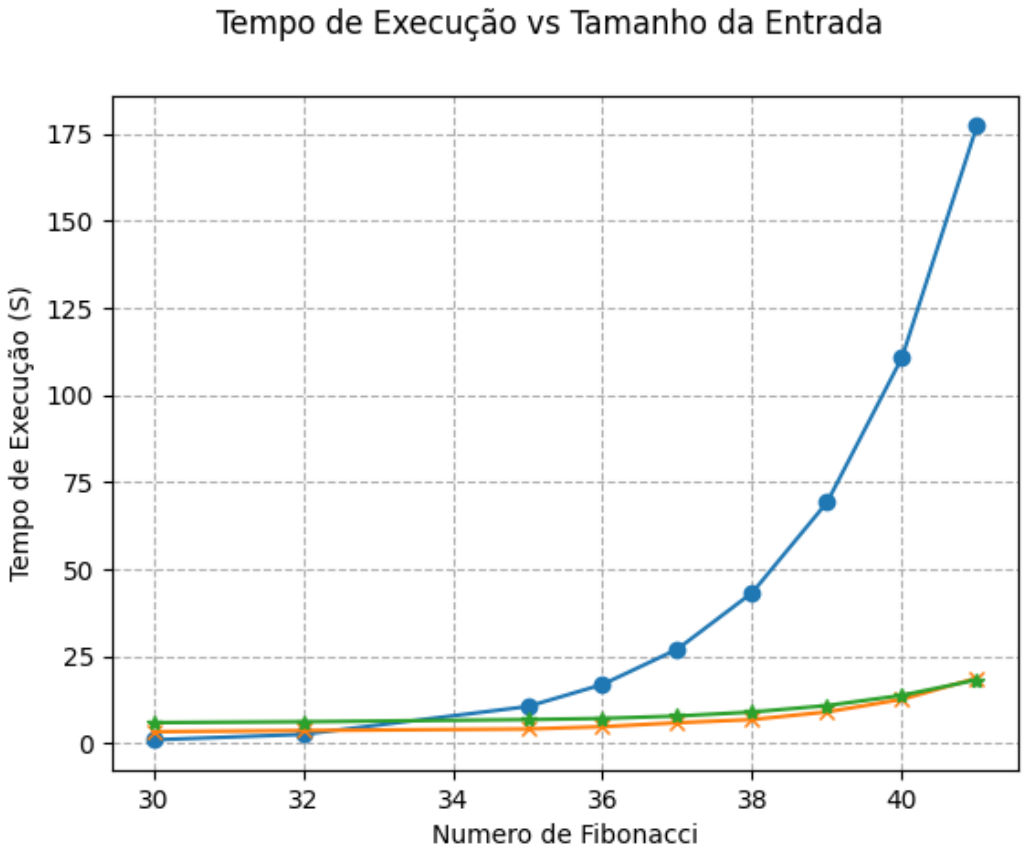

Figura 4.11. Comparação dos tempos de execução obtidos em simulação com Arduino Uno rodando a aplicação localmente (linha azul), no servidor da nuvem (linha verde) e no servidor da *cloudlet* (linha alaranjada).

a aplicação localmente, na nuvem e na *cloudlet* utilizando o Arduino Uno.

Os tempos de execução da aplicação nos servidores remotos foram muito parecidos com os tempos de execução nos mesmos servidores no experimento em ambiente real. Isso era esperado, já que os servidores têm as mesmas características em ambos os testes e o Arduino fica encarregado apenas de enviar os dados aos servidores. Como a quantidade de dados enviados é pequena (um número inteiro) as limitações do Arduino não causam grande diferença quando comparada ao Raspberry.

Esse resultado mostra como o MLOOF pode fazer uma diferença ainda maior no tempo de execução de tarefas quando os dispositivos são mais limitados.

4.4.7 Segundo teste: Escalabilidade

Para o segundo teste, com foco em escalabilidade, aumentamos o número de dispositivos mantendo a mesma quantidade de servidores.

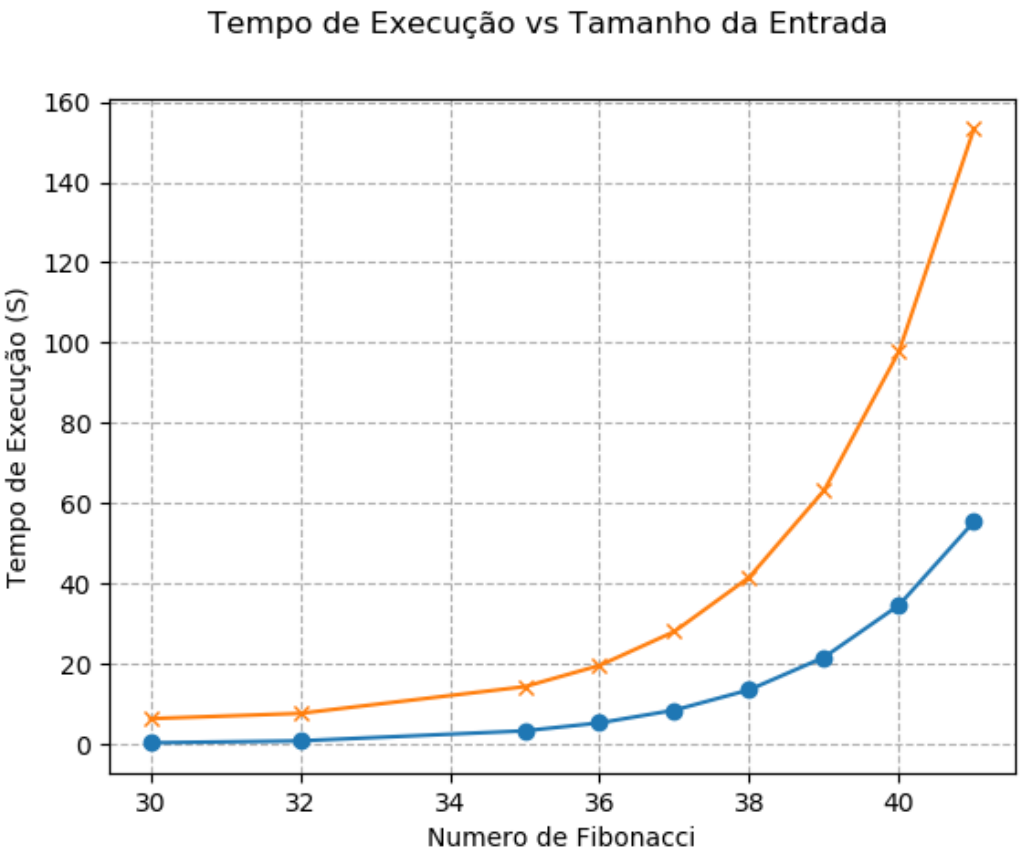

Figura 4.12. Comparação de resultados obtidos em simulação com Raspberry Pi (linha azul) e offloading para servidor no limite de sua capacidade de processamento (linha alaranjada).

Com o aumento da quantidade de dispositivos conectados em um único servidor, a utilização de CPU desse servidor aumenta proporcionalmente à quantidade de dispositivos e à carga que cada dispositivo adiciona ao servidor.

Enquanto o servidor possui capacidade de processamento livre, em outras palavras, enquanto a utilização de CPU do servidor não atinge seu limite, novos clientes se conectam ao servidor e executam o *offloading* com mesmo desempenho se comparado a um servidor que não possui nenhum cliente conectado. Quando a utilização de CPU chega a seu limite, o tempo de execução aumenta porque os recursos não são suficientes para atender todos os clientes. Nesta etapa da simulação, a utilização de rede não foi o gargalo do sistema.

Os dados podem ser observados no gráfico da Figura 4.12. O gráfico apresenta o tempo de execução de um cliente que, ao se conectar ao servidor, gera uma demanda de CPU maior do que a capacidade desse servidor, fazendo com que o tempo de resposta aumente.

A opção de realizar o *offloading* para o servidor sobrecarregado acarreta imensa perda de desempenho. Além do *overhead* gerado pela transmissão dos dados pela rede, um servidor com pouca ou nenhuma margem de CPU disponível demora mais do que o Raspberry para realizar o cálculo dos números. Os únicos ganhos nesta situação seriam em economia na bateria do dispositivo.

4.4.8 Terceiro teste: Topologias

Para o terceiro teste foram feitas mudanças nas características da rede sendo elas topologia, latência e banda dos links. A qualidade da rede é aspecto fundamental levado em consideração pelo motor de decisão do MLOOF ao escolher entre realizar ou não o *offloading* para algum servidor remoto. Esse teste tem como objetivo avaliar o impacto da rede no desempenho do sistema.

Características de topologia como latência e banda impactam diretamente o tempo de transmissão dos dados entre cliente e servidor, o que impacta a eficiência do processo de *offloading*. A latência adiciona tempo de espera a cada realização de *offloading* e a banda adiciona tempo caso a demanda seja maior do que a banda disponível.

Do ponto de vista do arcabouço de *offloading*, essas características da rede não podem ser controladas, apenas monitoradas, e elas tornam a opção de realizar o *offloading* mais ou menos atrativa quando comparada à execução local.

Para os experimentos, testes com variação de latência e largura de banda das conexões entre cliente e servidores foram realizados. Nos primeiros experimentos o aumento da latência e largura de banda de todas as conexões foram feitas de forma uniforme. A partir de certo valor de latência, a realização do *offloading* passa a ser menos interessante do que a execução do código no próprio dispositivo. O mesmo acontece ao diminuir a largura de banda abaixo de certo valor.

4.5 Conclusão

Este capítulo apresentou a avaliação do arcabouço de *offloading* MLOOF. O objetivo da avaliação é verificar se o MLOOF ajuda a melhorar o tempo de processamento das aplicações e a diminuir o consumo energético dos dispositivos, bem como quantificar esses ganhos. Para isso foram realizados experimentos com hardware real e executadas simulações.

Com o primeiro experimento constatamos a eficiência do MLOOF ao utilizá-lo para executar aplicações em um Raspberry Pi 3. Os dados apontaram grandes ganhos

em tempo de execução e economia de energia. Como a aplicação teste possui complexidade assintótica exponencial, os ganhos em tempo e de energia ficam cada vez maiores quando aumentamos o tamanho da entrada, já que os servidores possuem muito mais capacidade de processamento do que o Raspberry e a quantidade de energia consumida durante o *offloading* é constante. Os ganhos de tempo ao optar pelo *offloading* para a *cloudlet* foram ainda mais acentuados quando a aplicação realizava várias requisições de *offloading* seguidas, o que torna a opção pelo servidor da nuvem muito custosa. Os experimentos com *testbed* e servidor dedicado na nuvem comprovaram novamente os resultados obtidos no primeiro experimento.

Os objetivos dos experimentos simulados eram avaliar o desempenho do sistema em condições diversas que seriam muito difíceis ou impossíveis de serem avaliadas em ambiente real. Utilizando um simulador podemos variar as características de rede, carga de trabalho e características dos dispositivos. Os dados obtidos pelo simulador são, de certa forma, bem determinísticos, gerando resultados muito similares em simulações com os mesmos parâmetros. Por esse motivo os gráficos dos experimentos simulados não possuem anotação de margem de erro.

Essa flexibilidade na simulação nos permite avaliar o MLOOF em dispositivos IoT diferentes aos utilizados nos experimentos em ambiente real. O dispositivo escolhido para a simulação foi o Arduino Uno, que é um dispositivo com menos recursos computacionais se comparado ao Raspberry Pi utilizado nos testes com hardware físico.

As simulações mostram que os ganhos ao utilizar o MLOOF em dispositivos mais limitados são similares aos ganhos verificados no experimento real. Ademais, conseguimos ganhos muito maiores quando os recursos necessários para a execução da aplicação são maiores do que o dispositivo possui.

Capítulo 5

Conclusão

Este trabalho de dissertação de mestrado apresentou o sistema MLOOF, uma proposta de estratégia para descarregamento multi-nível de carga de dispositivos IoT e celulares, como uma evolução de trabalhos da literatura. A partir de dados históricos sobre a execução de determinada tarefa, bem como de dados de contexto, um motor de decisão situado em um dispositivo decide onde é o melhor local para executar a tarefa, se localmente, no servidor na *cloudlet* ou no servidor na nuvem.

A primeira contribuição do trabalho foi a concepção da arquitetura desejada e sua implementação. A implementação envolveu a criação de código que executa nos dispositivos, nas *cloudlets* e em servidores na nuvem que gerenciam o processo de *offloading* de forma automática.

A segunda contribuição, e um grande objetivo do trabalho, foi a avaliação da solução desenvolvida em experimentos reais e simulados. Resultados iniciais mostraram que a estratégia para descarregamento em três níveis (dispositivo, *cloudlet* e nuvem) traz resultados promissores em termos de redução de tempo de execução das tarefas ou consumo de energia dos dispositivos. Em todos os experimentos o MLOOF atingiu seus dois objetivos simultaneamente, diminuindo o tempo de execução das aplicações e o consumo energético no dispositivo. Experimentos em ambiente real utilizando a infraestrutura de um *testbed* comprovaram os ganhos obtidos no primeiro teste. Experimentos simulados permitiram verificar que esses ganhos se aplicam em situações nas quais são utilizados dispositivos menos potentes, servidores sobrecarregados e infraestrutura de rede menos adequadas.

Um objetivo secundário do trabalho era estudar melhorias para o motor de decisão utilizado, que foi herdado do trabalho utilizado como base para o MLOOF, o ULOOF. O motor apresenta boas predições mas é baseado em um histórico de execuções que deve ser populado ao longo da execução das aplicações para obter tais

predições. Isso significa que as primeiras decisões serão provavelmente piores do que as demais. Durante o trabalho foi desenvolvido um algoritmo de decisão distribuído, no qual os servidores na nuvem armazenam os dados de execução enviados pelos dispositivos clientes e criam regras básicas para tomada de decisão. Essas regras seriam úteis principalmente durante as primeiras tentativas de *offloading*, enquanto o histórico de execuções do dispositivo não possuir informações suficientes para realizar boas predições. Esse algoritmo não obteve bons resultados, não sendo muito eficiente para criar regras genéricas que consigam melhorar as predições e adicionava *overhead* aos dispositivos. Aperfeiçoamentos no motor de decisão, como exemplo a utilização de outro motor baseado em aprendizado de máquina, é um trabalho futuro.

A implementação do MLOOF foi feita em Java, o que permite sua utilização em uma gama de dispositivos compatíveis após a modificação de códigos específicos ao dispositivo em questão. Dispositivos IoT menores, porém, muitas vezes não executam código Java por isso não são compatíveis com o MLOOF. Deixamos para trabalhos futuros a adaptação do MLOOF para que possa ser utilizado em dispositivos sem suporte a código Java.

Outro possível trabalho futuro seria expandir a hierarquia em mais níveis, adicionando um mini data center como quarto nível entre os servidores da *cloudlet* e os servidores na nuvem. Ao adicionar mais níveis de comunicação podemos aumentar o *overhead* do sistema como um todo, mas podemos obter melhorias em certas aplicações que necessitem maior capacidade de hardware e evitaria a comunicação direta com os servidores na nuvem.

Para finalizar, agradecimentos especiais às agências CAPES, CNPq, Fapemig e ao projeto UNICOM: Modelos, Algoritmos, Protocolos e Técnicas de Comunicação sem Fio para a Nova Internet: de Cidades Inteligentes a Internet das Coisas via Comunicação Máquina-a-Máquina (processo CNPq 424307/2016-2), pelo apoio, seja na forma de bolsas de estudo ou auxílio a pesquisa.

Referências Bibliográficas

Al-Khafaji, H. M. R.; Alomari, E. S. & Majdi, H. S. (2019). Secured environment for cloud integrated fog and mist architecture. In *2019 IEEE International Conference on Electrical Engineering and Photonics (EExPolytech)*, pp. 112--116. IEEE.

Atzori, L.; Iera, A. & Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787--2805.

Bhalla, M. R. & Bhalla, A. V. (2010). Generations of mobile wireless technology: A survey. *International Journal of Computer Applications*, 5(4).

Chen, M. & Hao, Y. (2018). Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 36(3):587--597.

Cheng, N.; Lyu, F.; Quan, W.; Zhou, C.; He, H.; Shi, W. & Shen, X. (2019). Space/aerial-assisted computing offloading for iot applications: A learning-based approach. *IEEE Journal on Selected Areas in Communications*, 37(5):1117--1129.

Chun, B.-G.; Ihm, S.; Maniatis, P.; Naik, M. & Patti, A. (2011). Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pp. 301--314. ACM.

Cisco Systems, I. (2020). Cisco Annual Internet Report (2018–2023) White Paper. `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`. [Online; acessado em junho de 2020].

Cuervo, E.; Balasubramanian, A.; Cho, D.-k.; Wolman, A.; Saroiu, S.; Chandra, R. & Bahl, P. (2010). Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49--62. ACM.

El Baz, D. (2014). Iot and the need for high performance computing. In *Identification, Information and Knowledge in the Internet of Things (IIKI), 2014 International Conference on*, pp. 1--6. IEEE.

Flores, H.; Sharma, R.; Ferreira, D.; Kostakos, V.; Manner, J.; Tarkoma, S.; Hui, P. & Li, Y. (2017). Social-aware hybrid mobile offloading. *Pervasive and Mobile Computing*, 36:25--43.

Kemp, R.; Palmer, N.; Kielmann, T. & Bal, H. E. (2010). Cuckoo: A computation offloading framework for smartphones. In *MobiCASE*, pp. 59--79. Springer.

Kosta, S.; Aucinas, A.; Hui, P.; Mortier, R. & Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Infocom, 2012 Proceedings IEEE*, pp. 945--953. IEEE.

Kumar, K.; Liu, J.; Lu, Y.-H. & Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129--140.

Kumar, K. & Lu, Y.-H. (2010). Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51--56.

Lagar-Cavilla, H. A.; Tolia, N.; De Lara, E.; Satyanarayanan, M. & O'Hallaron, D. (2007). Interactive resource-intensive applications made easy. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, pp. 143--163. Springer-Verlag New York, Inc.

Mahmud, R. & Buyya, R. (2019). Modelling and simulation of fog and edge computing environments using ifogsim toolkit. *Fog and edge computing: Principles and paradigms*, pp. 1--35.

Mao, Y.; Zhang, J. & Letaief, K. B. (2016). Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590--3605.

Mell, P.; Grance, T. et al. (2011). The nist definition of cloud computing.

Milinković, A.; Milinković, S. & Lazić, L. (2015). Choosing the right rtos for iot platform. In *Proceedings of the international scientific professional symposium Infoteh, Jahorina*, pp. 18--20.

Miorandi, D.; Sicari, S.; De Pellegrini, F. & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497--1516.

Motlagh, N. H.; Bagaa, M. & Taleb, T. (2017). Uav-based iot platform: A crowd surveillance use case. *IEEE Communications Magazine*, 55(2):128--134.

Neto, J. L. D. (2016). Uloof user-level online offloading framework: user-level online offloading framework.

Neto, J. L. D.; Yu, S.-y.; Macedo, D. F.; Nogueira, J. M. S.; Langar, R. & Secci, S. (2018). Uloof: a user level online offloading framework for mobile edge computing. *IEEE Transactions on Mobile Computing*.

Persson, P. & Angelsmark, O. (2015). Calvin-merging cloud and iot. In *ANT/SEIT*, pp. 210--217.

Qasem, M. H.; Abu-Srhan, A.; Natoureah, H. & Alzaghoul, E. (2020). Fog computing framework for smart city design. *International Journal of Interactive Mobile Technologies (iJIM)*, 14(01):109--125.

Satyanarayanan, M. (1996). Fundamental challenges in mobile computing. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pp. 1--7. ACM.

Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1):30--39.

Satyanarayanan, M.; Bahl, P.; Caceres, R. & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4).

Shaukat, U.; Ahmed, E.; Anwar, Z. & Xia, F. (2016). Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges. *Journal of Network and Computer Applications*, 62:18--40.

Shi, C.; Habak, K.; Pandurangan, P.; Ammar, M.; Naik, M. & Zegura, E. (2014). Cosmos: computation offloading as a service for mobile devices. In *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pp. 287--296. ACM.

Toor, A.; ul Islam, S.; Ahmed, G.; Jabbar, S.; Khalid, S. & Sharif, A. M. (2019). Energy efficient edge-of-things. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):82.

UFMG, D. (2020a). FUTEBOL UFMG. `http://futebol.dcc.ufmg.br/`. [Online; acessado em março de 2020].

UFMG, D. (2020b). TUTORIALS - FUTEBOL UFMG. `http://futebol.dcc.ufmg.br/tutorials_jfed.html`. [Online; acessado em março de 2020].

Verbelen, T.; Simoens, P.; De Turck, F. & Dhoedt, B. (2012). Aiolos: Middleware for improving mobile application performance through cyber foraging. *Journal of Systems and Software*, 85(11):2629--2639.